DOCTORAL THESIS

Artificial Neural Networks with Function Point Analysis for Software Development Effort Estimation.

Wittig, Gerhard

Award date:
1995

School of Information Technology

Bond University

Artificial Neural Networks with Function Point Analysis for Software Development Effort Estimation

by

Gerhard Eduard Wittig

Submitted to Bond University in fulfilment of the requirements for the degree

Doctor of Philosophy

January 1995

Abstract

Reliable prediction of system size and development effort is a necessary prerequisite to effective project planning and control. Using artificial neural networks this thesis extends recent research in software development effort estimation.

Progress in effort estimation has been poor and accuracy has been disappointing. This is not attributable to a lack of effort or intellectual capability, but is an indication of the complexity of the problem. Interrelationships between various factors affecting development effort are complex, not fully understood, and have made development cost estimation difficult and sometimes inaccurate.

The value of neural network modelling techniques in performing complicated pattern recognition and non-linear estimation tasks has been demonstrated across an impressive spectrum of applications. This thesis reports on the ability and limitations of artificial neural networks in recognising and modelling the complex patterns of interrelationships between software development attributes and project effort.

The artificial neural network issues of network architecture and topology, various parameter settings and scaling techniques, and the problems of generalisation and overfitting are addressed in the development of effort estimation models. Generally cascade networks with their ability to dynamically develop the near optimum network topology are used.

Limitations of project data were identified and the model development and analyses were conducted within these constraints. To assess the neural networks' capability they were tested across several datasets. In addition to testing their ability of approximating a measurable function from one finite space to another, the networks' estimation capability was also assessed with limited project data in the presence of noise and with few observations.

A large set of simulated project data was developed to overcome the problem of limited observations. Cascade networks were assessed for their ability to accurately

estimate development effort by modelling several development attributes which were responsible for a large range in development productivity. The effect on estimation accuracy of different size measures, as well as the inclusion of various development attributes is also tested on smaller datasets.

The ability of neural networks to accurately estimate development effort using the internationally recognised Australian Software Metrics Association project data was assessed. The effect of the inclusion of several cost drivers into the neural network model for improved estimation accuracy is demonstrated.

Advances in effort estimation are likely to be a slow and gradual process. This research project is seen as part of that long and difficult path.

Declaration

This thesis represents my own work and contains no material which has been previously submitted for a degree or diploma at this University or any other institution, except where due acknowledgment is made.

Signature 

Witness 

Date 2.07.95

Additional Publications

The following is a list of publications by the candidate on matters relating to the thesis.

Journal

1. G.R. Finnie, G.E. Wittig, and D. Petkov, Prioritization of Factors Affecting Productivity of Software Development using the Analytic Hierarchy Process, *Journal of Systems and Software* , vol. 22, no. 2, pp 129–139, 1993.

2. G.E. Wittig, and G.R. Finnie, Using Artificial Neural Networks and Function Points to Estimate 4GL Software Development Effort, *Australian Journal of Information Systems*, vol. 1, no. 2, pp 87–94, 1994.

3. G.R. Finnie, and G.E Wittig, The Effect of System and Team Size on 4GL Development Productivity, *South African Computer Journal,* no. 11, pp 18–25, May 1994.

Conference

1. C.N.W. Tan, and G.E. Wittig, The Study of the Parametric Effect on an Experimental Backpropagation Model, *New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems* , November 1993.

2. C.N.W. Tan, and G.E. Wittig, Parametric Variation Experimentation on a Backpropagation Stock Price Prediction Model, *Australian and New Zealand Conference on Intelligent Information Systems ANZIIS-93,* December 1993.

3. G.E. Wittig, Artificial Neural Networks for 4GL Software Development Effort Estimation and Analysis, *International Federation for Information Processing Conference–IFIP TC8AUS Doctoral Consortium*, May 1994.

4. G.E. Wittig, and G.R. Finnie, Software Design for the Automation of Unadjusted Function Point Counting, *International Federation for Information Processing Conference–IFIP TC8AUS Conference*, May 1994.

5. G.R. Finnie, and G.E. Wittig, Reverse Engineering for Calibrating Software Development Metrics, *5th Australian Conference on Information Systems*, vol. 2, pp711–718, Sept 1994.

Working Papers

1. C.N.W. Tan, and G.E. Wittig, *The Study of the Parameters Effect of a Backpropagation Stock Price Prediction Model*, School of Information Technology, Bond University Working Paper 1993-3-091/B, May 1993.

2. G.E. Wittig, and G.R. Finnie, *Estimating 4GL Software Development Effort Using a Backpropagation Network*, School of Information Technology, Bond University Working Paper 1993-3-103/B, November 1993.

3. G.E. Wittig, and G.R. Finnie, *The Partial Automation of Function Point Counting,* School of Information Technology, Bond University Working Paper 1993-3-104B, November 1993.

4. G.E. Wittig, and G.R. Finnie, *Software Development Productivity Variations in Function Point Research Data,* School of Information Technology, Bond University Working Paper 1994-3-108B, April 1994.

5. G.E. Wittig, and G.R. Finnie, *A Study of Software Development Effort Estimation Model Performance*, School of Information Technology, Bond University Working Paper 1994-3-109B, April 1994.

Acknowledgments

This thesis was supervised by Associate Professor Gavin Finnie of Bond University. I thank him for accepting this task and pay tribute to the high quality of his supervision. Gavin's guiding influence and considered comments throughout this research project are gratefully acknowledged.

I thank Bond University for accepting me as doctoral candidate and for awarding me a full scholarship for the duration of this thesis. The high standards set by this university provided a role model which has been an encouragement to emulate. The good facilities provided by Bond University are also acknowledged, without which it would have been considerably more difficult to complete this thesis.

My gratitude is also extended to Dr Mats Lundeberg of the Stockholm School of Economics, for his advice regarding the planning and implementation of this thesis project. This provided the necessary focus and motivation to ensure its completion.

A key aspect of this thesis has been the use of cascade networks. I thank Dr Scott Fahlman, Principal Research Scientist of Carnegie Mellon University for providing the software which he had developed and for the assistance and valuable advice given. I also gratefully acknowledge the support in the use of this tool given by Professor Joachim Diederich and Thomas Szabo from the Queensland University of Technology. Their assistance has been invaluable. Joachim has also provided guidance and advice in the use of artificial neural networks generally and I thank him for this. Thomas developed the k-Nearest-Neighbour (k-NN) software which was also used in this thesis, and I thank him for permitting its use for this research project and also for his valuable advice on several cascade network and k-NN issues.

The contribution of numerous other people is also gratefully acknowledged. These include:

- Dr Barbara Kitchenham of the National Computing Centre in Manchester UK for project data
- Capers Jones of Software Productivity Research, Burlington, USA, for the SPQR/20 project estimation software
- Pam Morris from Total Metrics, Melbourne, for her discussions and knowledgeable insight into function point counting
- Professor George Lendaris from Portland State University, USA, for discussing and providing advice on several artificial neural network problems
- Paul Pyyvaara from ITS, Bond University, for his assistance in setting up the cascade network software on the Sun workstation
- Jill Denholm, Bond University library, for the time spent conducting literature surveys and locating some research publications

Dedication

This thesis is dedicated to my family. To my wife Janet, and my children Eduard and Megan for their continued loving support and understanding over the years.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

The significance of software measurement and estimation is summarised by Jones [56] as follows:

> *Measurement is the basis of all science, engineering, and business. Planning and estimation are mirror images of measurement. The factors and metrics that were recorded during project development are aimed toward the future of uncompleted projects.*

Reliable prediction of size and effort in software development projects is a necessary prerequisite to developing reliable cost and schedule estimates [18]. The size and development effort measures, such as function points or lines of code developed per person-month, act as technical productivity and performance indicators that facilitate the tracking and control of software developments.

System size forms the basis of estimation models [16]. The size of a system is included with a productivity function to estimate the amount of effort required to develop a system. This productivity function is dependent on many attributes in the development environment causing variations in productivity [16]. If there were no productivity variations it would be a simple matter of estimating effort by applying a common

adjustment to all size measures. Empirical investigations reveal large software development productivity variations in commercially developed systems [123]. To model the productivity function requires the quantification and modelling of the interrelationships between the various development environment attributes. Interrelationships between the various factors affecting development effort are complex, not fully understood [63] and have made development cost estimation difficult and sometimes inaccurate [31] [40] [50] [60] [66] [78].

Total world software cost was estimated to have been $US250 billion in 1990 [13]. With this magnitude of software investment, software project estimates have to be carefully considered as they have a direct and significant impact on the efficiency and effectiveness with which software is developed [78].

An over-estimation of software size results in an inflated impression of their value and the development team's productivity. This implies that inefficient developments are not recognised as such, with no appropriate action being taken. On the other hand viable projects with real potential may be rejected as 'too expensive', resulting in an opportunity cost to the firm.

Up to 15 percent of new development projects are abandoned before completion, largely due to their development cost being initially underestimated [58]. This usually represents a total waste of expenditure on those developments. Under-estimation also results in some projects not being fully completed due to time constraints. Developers may omit important features or system testing, resulting in incomplete and unreliable systems [60].

It is by no means easy to estimate development effort accurately and much effort has been expended to develop metrics which attempt to measure size and complexity of programs and systems [16].

The value of neural network modelling techniques in performing complicated pattern recognition and non-linear estimation tasks has been demonstrated across an impressive spectrum of applications [116]. For this thesis their usefulness in metrics models has been investigated. A survey [95] shows that the recent growth of neural network applications has been quite remarkable. It was just four years ago that the

only widely reported commercial application outside the financial industry was the airport baggage explosive detection system [44].

Since that time scores of industrial and commercial applications have become known. A few of these applications include telecommunications, particle accelerator beam control, credit card fraud detection, machine and hand-printed character recognition, cursive handwriting recognition, mass spectra classification, quality control in manufacturing, petroleum exploration, war on drugs, medical applications, financial forecasting and portfolio management, and loan approval. The details of many are considered corporate property and shrouded in secrecy. This growth in neural network applications is in part due to the availability of an increasingly wide array of dedicated neural network hardware, in the form of accelerator cards for PC's and workstations. Complementing the hardware are scores of commercial software packages [95].

Artificial neural networks (ANNs) are recognised for their ability to provide good results when dealing with problems where there are complex relationships between inputs and outputs, and where the input data is distorted by high noise levels [92]. The software development environment from which development effort estimates are generated, are characterised by these attributes.

1.2 Importance of this Research

Information systems development is an important component of commerce and industry. Boehm and Papaccio have estimated that in the USA the software costs are approximately 2 percent of their Gross National Product [13]. The value of information is being recognised in the current business environment. Not only are information systems being used at the operational level, but they are increasingly being used as strategic tools [41] [81] [96]. The use of information technology to gain competitive advantage has been well documented. In most large organisations it is virtually inconceivable to consider operating without massive information systems support.

The competitive business environment demands that organisations employ information technology, and the above figures suggest that it is a major cost item in company

budgets. The efficient running of an organisation is vital to its existence. This applies not only to its operational activities, but equally to information systems development and maintenance. To ensure this efficiency requires effective planning and control of these projects.

Both the planning and control function require the capability to accurately and reliably measure the software being developed [61]. In information systems development the planning function emphasises estimation of the size of systems in order that appropriate budgets and schedules can be determined. Without a reliable size estimate this process is likely to become highly inaccurate [61]. Implicit in this statement is the assumption of an estimate of development effort from the system's size measure. Knowledge of a systems size alone does not permit the generation of budgets and schedules. Effort estimation therefore is a key issue in this process.

Control of software projects necessitates a means to accurately measure progress of a project. A further need for accurate size measures is to conduct evaluations after the completion of a project to establish development productivity. This also allows for evaluations on tools and techniques employed on the project to improve efficiency. Again as above an effort estimate is required to permit a comparison of this against the actual development effort at the completion of the project.

Reliable software size measurement is thus a crucial aspect of the management of an important aspect of an organisation's business, which not only consumes large resources, but also has strategic implications. The key factor which facilitates the utility of system size for management functions is the estimation of development effort.

According to Tate [109] progress in effort estimation from system size has been poor over the last 10 to 15 years, and accuracy has been disappointing. This is reflected in a review of estimation model performance and is discussed in more detail in Chapter 2. This apparently poor performance of metric models should not be attributed to a lack of effort or intellectual ability, but rather as an indication of the complexity of the problem.

The importance of this study is the assessment of the potential of artificial neural networks, which have shown exciting promise in many other applications, for the complex problem of software effort estimation. Any advances in effort estimation is

likely to be a slow and gradual process, and this research project is seen as part of that long and difficult path. Any improvement in effort estimation will eventually benefit this important industry as a whole, and justifies the effort devoted to this topic.

1.3 Thesis Organisation

This chapter serves as introduction, giving a brief overview of some of the issues as well as briefly describing the importance of this research project. The next chapter discusses some of the prior research which was conducted as preparation to the project.

Apart from an overview of metrics generally, and Function Point Analysis in particular, many of the areas relevant to the topic are discussed. Included in this discussion is a brief review of various function-based metrics, a review of various studies on the effect of development attributes, and the reliability of the function point measurement. This section covers a discussion on measurement theory, on the function point model adequacy, as well as a review of inter-rater inconsistency studies. The next section in this chapter reviews the productivity variations which are inherent in development projects. The larger this variability is for similar systems, the larger the adjustment or explanation that is required. A greater variability in productivity implies a greater influence of cost drivers, and the greater this influence the more complex the model is likely to be. The final section of this chapter reviews the published results of estimation model performance.

Chapter 3 also covers prior research which was conducted for this thesis, and deals with various aspects of artificial neural networks. Some neural network background information is included in Appendix B. Artificial neural networks are models of the biological neural structures and are an attempt to exploit some of the human brain's potential. The first section in Chapter 3 briefly discusses the back-propagation network learning mechanism. The next section reviews some issues of network topology, architecture, and parameter settings with particular reference to back-propagation networks. The generalisation ability of neural networks is an important aspect. The next section discusses this and the problem of over-fitting.

In the final sections of this chapter cascade networks are discussed, and in particular their architecture and operations, as well as parameter settings. Also reviewed are the advantages of these networks with particular reference to their use in the complex software development environment.

The next chapter discusses the research methodology. In addition to the general and specific research design, the research population, the analysis procedures, the tools used and network parameter settings are discussed. In this chapter the pilot study which was conducted to assess the feasibility of this research project is described.

Chapter 5 reports on the analysis and results. The analysis is divided into six reasonably autonomous research projects. Some specific research design procedures, the implementation of the experiments, as well as the analysis of the various results is given in six sections of this chapter. Also cascade network stability and a comparison to back-propagation models is discussed.

The next chapter includes a brief description of the software program that was developed. To use cascade networks for effort estimation requires the use of a spreadsheet and a word processor program to manipulate the data into a suitable form for input into the neural network. These same tools are used to present the network output into a more useable form. Cascade2, the cascade network used for this research project is not a very user-friendly tool, as the control of the program is done partly through the development and use of a network header file. Unless a user is familiar with some of the technical aspects, it may be a little difficult to use this program. To try and overcome this a program was developed which automates many of these tasks, with a user-friendly interface for ease of use. Some of these aspects are discussed in this chapter.

In the last chapter the conclusions and implication of results are discussed, as well as recommendations for further research.

To avoid including in the main research description supportive and background information or large tables, these were placed in the Appendices. Appendix A contains details of the technique of function point analysis, Appendix B details of back-propagation neural networks, Appendix C the productivity adjustment factors of the

Mermaid2 project data, Appendix D the outputs of three analyses, and Appendix E the software code of the program mentioned above.

1.4 Research Question

Conte, Dunsmore, and Shen [16] state that for an adequate effort estimation model the mean of the relative errors should be less than 0.25, and at least 75 percent of the predicted values should fall within 25 percent of their actual observations. Other researchers [75] have accepted these measures and have assessed the adequacy of the models they have developed according to this set of criteria.

In general the objective of this research project is to assess the potential of artificial neural networks for development effort estimation. The software development environment is complex and effort estimation is difficult. Neural networks on the other hand have indicated their potential for complex pattern recognition, even in the presence of relatively high noise levels, and research has indicated that they are universal approximators and are capable of approximating any measurable function to any desired degree of accuracy. This implies that any lack of success in applications must arise from inadequate learning, insufficient numbers of hidden units, or a lack of a deterministic relationship between input and target [46].

To achieve the objective of assessing the adequacy of artificial neural network effort estimation models the following research question was developed:

Are artificial neural networks capable of providing an adequate development effort estimation model in which 75 percent of the predicted values are within 25 percent of the actual observations, and where the mean absolute relative error is less than 0.25?

1.5 Limitations of the study

The development of an effort estimation model is dependant on the tools and techniques used, as well as the data on which the model was developed. Sufficient reliable software development project data is not readily available, and project

databases measured by artificial neural network requirements for adequate generalisation [10] are relatively small.

This necessitated that the neural network models be developed within the reality of this restriction. It prevented the full potential of the neural network models being exploited. Both the number of observations in the training and testing sets, as well as the availability of only a limited number of project attributes contributed to this.

Artificial neural network technology is continually being developed and it appears inevitable that current models will be superseded by improved versions. This research project was conducted using current and research neural network tools, and this thesis has identified some aspects which may need further research effort. The lack of understanding of some aspects of artificial neural networks, and in particular the incomplete theory on parameter settings and optimum topology selection necessitates a partially experimentative approach to developing the models. This limits their development and use as general purpose management tools. The lack of a definitive theory in some of these areas also implies that neural network performance is partially dependent on the skill of the developer.

The results obtained in this research project thus have to be analysed and interpreted within the above limitations.

1.6 Summary

Effective software development requires accurate estimation of project effort for effective project management. This research project assesses the capability of a new approach by using artificial neural networks to develop an adequate model to generate project effort estimates.

Estimation issues such as the characteristics of project data and the performance of current estimation models are considered. The underlying principles and capabilities of neural networks are discussed, especially their complex pattern recognition capability. To evaluate the ability of artificial neural networks to estimate development effort accurately they are assessed in six sets of research projects to test a wide range of estimation aspects. The final research project includes the assessment of a model

developed on the Australian Software Metrics Association project data, and the evaluation of the effort estimation capability of this model.

Despite the successful development of an adequate estimation model, some further areas are identified for research in the continuous quest for improved software development effort estimation.

Chapter 2

Significant Prior Research

2.1 Introduction

Preparation for this research project included a review and analysis of several research publications. The focus of this project is the use of artificial neural networks for software development effort estimation and productivity analysis. This necessitated a review of the concepts of neural networks, as well as familiarisation with recent technological advances of those aspects of neural networks which are useful for effort estimation. These aspects are covered in Chapter 3.

A review of metrics research in the software development environment was conducted in preparation of the research design. Relatively large project datasets are required for neural network research to be conducted satisfactorily. To gather this type of information the metric should be a reliable, inexpensive and preferably automated measure. A historical overview of metrics, with special focus on the function point metrics was conducted and is discussed in Section 2.2. The reasons why the function point metric was selected as a size measure for this thesis is discussed in this section. This is followed by a brief overview of metrics, and in particular the review of nine function-based methods to highlight their respective advantages and disadvantages.

The reliability of the function point measurement was then reviewed and analysed, and is discussed in Section 2.3. Unreliable inputs introduce noise into a model, and this

section reviews some potential sources of noise in research data. Various aspects were considered. Initially the problem was viewed from the measurement theory perspective, followed by an investigation into the function point model adequacy and some criticism of published models. Some of the models discussed here have been included in research aspects of this thesis in Chapter 5, and this section serves as a caveat in the interpretation of the analyses and results. Some practical issues were then reviewed, and in particular measurement consistency, as well as some suggested variations and improvements of the function point metric.

An analysis of software development productivity revealed the magnitude of the variability. The size of this problem and its implications are discussed in Section 2.4. This variation in development productivity is largely the result of the development project and environmental differences, which manifest themselves in the cost drivers. Research which has been conducted on the effect, interrelationship and magnitude of some cost drivers is also reviewed in this chapter. The approach of other researchers, as well as an understanding of some of the problems they encountered, assisted in formulating the approach taken by this research project.

The published research results of the effect of the various development attributes on effort do not yield universal agreement. In Section 2.5 several research publications on this topic are reviewed and in some cases further analyses were conducted. This section not only indicates some of the cost drivers which researchers have considered to have a significant impact on development effort, but also highlights the difficulty in interpreting project data.

A study was made of various estimation issues, and in particular the various factors which affect estimation model performance. To determine the current status of estimation models, their performance was reviewed and the results are reported in Section 2.6. The generally disappointing results are a reflection of the complexity of the problem, especially when consideration is taken of the large resources and intellectual focus which have been deployed in this endeavour. Numerous highly capable researchers and practitioners have applied their minds to this problem, but progress has been slow on this long road to finding a satisfactory solution.

Having made the decision to use function point measures as the main unit of measure, the Function Point Analysis (FPA) methodology was reviewed. FPA is referred to on several occasions in the chapters reporting on the research design and analysis aspects. Where knowledge of some detail of the FPA approach is required, reference is made to this detail in Appendix A.

Some research results, where the number of observations are relatively large, and where the individual details are not critical to their interpretation, have been placed in Appendix D. This tends to improve the readability of the results, as large tables in the text may in some cases be slightly disruptive. Some detail of the mechanics of back-propagation artificial neural networks, as well as some other research details have for similar reasons been included in the appendices.

The topics covered in this and the next chapter are fundamental to and were influential in determining the research design and guided the analysis of results.

2.2 System Size Metric Review

The system size measure forms the basis of most metrics and productivity assessments. There are many possibilities of expressing the size of a program. Lines of code has in the past been the most popular software measure. A criticism is the lack of a universally accepted definition, which makes the comparison of much of the published research data very difficult.

Jones [57] has identified 11 line counting variations, which occur either at the program or project level. The difference between the most diffuse and the most compact technique varies the count by a factor of 5. Many authors do not define which counting convention was used, which implies a possible variability of 500 percent due to this alone.

Another difficulty with the lines of code measure is that inter-language comparisons are difficult. The ratio of source statements to executable statements varies considerably. For example spreadsheets may generate on average 50 executable statements per source statement, while Pascal has approximately 3.5, and C has 2.5

[57]. A comparison between a spreadsheet application line of code and say C is difficult and there is no agreement on how exactly this should be done [16].

A criticism of a lines of code measure within one language is that some applications may contain simple lines of code, while others may be complex lines of code, again making effort estimation productivity comparisons difficult. To accommodate this variance a scheme was proposed by Halstead [39] which uses tokens to weight lines according to the amount of 'content'. This scheme proposed a family of metrics commonly called the Software Science metrics.

Other suggestions were also made by McCabe [76] who proposed a logic structure metric commonly known as McCabe's Cyclocmatic Complexity, to reflect variances in complexity based on decision counts. Other logic structure metrics which have been proposed are minimum number of paths and reachability metrics [98], metrics reflecting nesting levels [73], and transfer usage metrics [16].

A further difficulty with lines of code which has practical implications, is that the system size can only be determined reasonably accurately when the system development has almost been completed. For effective planning and control of projects estimates early in the development life-cycle are necessary.

In 1979 Albrecht [1] departed completely from lines of code as a measure of size, when he proposed his Function Point Analysis based on the amount of functionality provided to the user by the system. This solved many of the problems associated with lines of code, but initially lacked a universally acceptable definition. The International Function Point Users Group (IFPUG) has now developed a comprehensive counting practices manual, with their version 4 [35] being available since January 1994. The IFPUG counting standard is widely accepted as one of the major standards of function point counting [55].

Function Point Analysis is also not without some perceived shortcomings and some of these are covered in more detail later in this chapter. To improve on the original function point counting technique various adaptations have been suggested [56], [89], [103]. MacDonell [73] identified and evaluated nine function-based assessment and estimation methods. These methods were:

1. Bang metrics [18]
2. Bang Metric Analysis (BMA) [14]
3. CASE size metrics [110] [111]
4. Entity metrics [37]
5. Function Point Analysis [1]
6. Information Engineering metrics [48]
7. Mark II FPA [103] [104]
8. Metrics Guided Methodology [86]
9. Useability measures [117]

All these approaches were assessed against the following six criteria.

A. Automatic – Can the method be automated, requiring no input from personnel?

B. Comprehensive – Are aspects of the size and inter-connectivity of the data, process and user interface representations considered by the model?

C. Objective – Will the model consistently produce the same result for a specific system, irrespective of the person performing the assessment?

D. Specification basis – Can the assessment reliably be done from the requirements specification, implying implementation independent system representations?

E. Tested – Has the model been tested using appropriate 'real-world' data?

F. Validated – Has the model been evaluated on systems other than those used for testing?

A summary of the results is given in Table 2.1, where the nine methods are listed and the evaluation criteria was either met, Y (yes), or otherwise, N (no).

Table 2.1 Summary of MacDonell Assessment

	Evaluation Criteria						Rating
Method	A	B	C	D	E	F	out of 6
1	N	Y	N	Y	Y	N	3
2	Y	Y	Y	Y	Y	N	5
3	Y	Y	Y	Y	Y	N	5
4	Y	Y	Y	Y	N	N	4
5	N	Y	N	N	Y	Y	3
6	N	Y	N	N	Y	Y	3
7	N	Y	N	Y	Y	Y	4
8	N	Y	N	N	N	N	1
9	N	N	N	Y	N	N	1

The three criteria for which the models rated lowest on average were automation, objectivity, and validation. This highlights some problem areas to which further research attention may have to be directed. Function Point counts are the main size metric used for this thesis, mainly because of its general acceptance in commercial developments [23] [40] [56]. The areas in which MacDonell [73] concluded that this method did not meet with the criteria are interesting and the problems which arise from these are discussed Section 2.3. In particular the lack of objectivity combined with the difficulty of automating the function point count require further research effort. The estimation model performance discussed in Section 2.6 may also be influenced by Function Point Analysis not meeting these criteria.

A brief summary of the relative merits of each method as assessed by MacDonell [73] using the above-mentioned criteria, is given in Table 2.2. Further research is needed to validate this assessment to determine whether these benefits are evident in empirical performance.

The brief review of metrics provides a summary of some issues which are of concern in effort estimation. Later sections of this chapter reflect how some of these attributes manifest themselves in aspects which may eventually lead to poor model performance. One metrics aspect which is retarding research progress is the unavailability of a widely

accepted automated functional size measurement. This contributes and leads to a shortage of software measurement [3] and results in restricted datasets to conduct research [59]. This section has reviewed some of the metrics options which are currently available.

Table 2.2 Assessment Comments (MacDonell [73])

Method	Comments
1	Intuitive and able to use early in the life-cycle, but the method is partly subjective and not validated
2	Easy to automate, contains no subjectivity, but has only been minimally tested and not adequately validated
3	The basis is in conceptual models, which have been tested but not adequately validated.
4	Early use in life-cycle and objective, but as yet untested
5	Question over objectivity, but is widely used, tested, and supported
6	Has several subjective elements, but is relatively comprehensive
7	Not completely objective or automatable, but is well tested
8	Partially automatable, but only after conceptual phase
9	Somewhat subjective and completely untested

2.3 Reliability of Function Point Measurement

2.3.1 Introduction

Software size is a critical component of development productivity and quality ratios. Source lines of code (SLOC) which in the past was a popular size measure has been criticised [57] and lost some of its popularity.

Function Point Analysis (FPA) has been suggested as an alternative [2] to resolve some of the weaknesses associated with SLOC. Function points have become a widely

accepted metric with both academics and practitioners [17] [23] [56]. Despite their wide use they have not been without criticism. Pressman [84] claims that FPA, like SLOC is relatively controversial with some opponents claiming that the method requires some 'sleight of hand', in that computation is based on subjective rather than objective data. Symons is critical of some of the fundamental issues of FPA and developed his Function Point MkII size metric as an alternative [103] [104]. Some of Symons' criticisms are reviewed in Section 2.3.2, while measurement theory in general is reviewed in Section 2.3.3. Matson, Barrett and Mellichamp are critical of the FPA model adequacy or the explanatory power of the independent variables in accounting for the variability of the dependent variable, as well as the model stability [75]. These issues are reviewed in Section 2.3.4. Another aspect of FPA is the inter-rater inconsistency [59]. The magnitude and potential causes are discussed in Section 2.3.5.

2.3.2 Function Points MkII

Symons [103] [104] in his critical review of FPA highlights the following difficulties.

- Classification of all system components or functions into only three categories of simple, average or complex, has the merit of being straightforward, but is an oversimplification of reality.
- Albrecht's choice of weights was determined by 'debate and trial'. Symons considers some more objective assessment of the weights as advisable.
- The way in which internal complexity is taken into account appears inadequate, especially in systems of very high internal complexity, where FPA tends to understate their size. In addition complexity is determined somewhat subjectively. The results of Gibson and Senn [36] show that subjectively based complexity assessments are unsatisfactory and they suggest a need for further research on the efficacy of subjectively based software metrics.
- The question of whether function points are summable, and to what extent they are independent or overlapping has not been resolved conclusively.
- The restriction of 14 factors included in the determination of the technical complexity adjustment (TCA) appears rather restrictive. Also with these factors

there appears to be some overlapping. Research conducted by Kitchenham [63] concluded that the original 14 technology factors are not independent and could be represented by six new factors.

- The weights of each of the 14 factors, which are restricted to the range of 0 to 5 are unlikely to be able to accommodate the full range of variation usually encountered in development. With all factors having the range 0 to 5, it means that if say two factors, for example reuseability and performance, are judged to influence the design to the same degree, they will contribute an equal weight coefficient to the problem size adjustment. Symons questions this equal weighting, and certainly other models, for example COCOMO, weight differently the various factors they take into account. This thesis project investigates some of these issues using neural networks (Sections 5.5.6, 5.6.4.3)

Symons in his effort to overcome these difficulties has proposed his MkII FP model [103]. The information processing size is expressed in unadjusted function points, but is now the sum of the weighted number of input data element types, the weighted number of entity-type references, and the weighted number of output data element types. Symons scaled the MkII weightings so that the average size for the 8 systems he examined, the unadjusted function points were the same for both Albrecht's FPA model [2] and the MkII model [104]. This then permitted a comparison of the two models to be made. The weightings suggested by Symons for information processing is thus:

$$UFPs = 0.44N_I + 1.67N_E + 0.38N_O$$

where N_I is the number of input data element types
N_E is the number of entity-type references
N_E is the number of output data element types

In addition Symons expanded the number of factors affecting the technical complexity from 14 to 20. Symons also questioned the weight of each degree of influence, and has suggested that this should vary with technology, and can for some applications be half of Albrecht's weighting. This would results in the following formula:

$$TCA = 0.65 + 0.005 \times DI$$

where DI = degrees of influence of the technical complexity factors

Research has not proved conclusively that Symons' MkII model is a better size metric enabling more accurate estimates to be made than when Albrecht's FPA model is used. Results of research done by Ratcliffe and Rollo [88] are inconclusive, but they did comment that in practice Symons' MkII version is more difficult to apply than Albrecht's model. Even Albrecht's model initially appears deceptively simple to apply, but is considerably more difficult to use in practice [104].

2.3.3 Measurement Theory

According to Fenton [30] the scientific measurement theory for software metrics has been largely ignored by both practitioners and researchers, and that much published work in software metrics is theoretically flawed.

Software metrics may be used either for assessment or prediction. Accurate prediction is dependent on the accurate measurement and quantification or classification of various attributes. In addition to these measures the predictive model requires the clear definition of the procedures for determining the model parameters and interpreting the results. The model together with these definitions is called the predictive system.

Two general types of measurement are identified.

- The direct measurement of an attribute is where this is not dependent on the measurement of any other attribute
- The indirect measurement of an attribute is where one or more other attributes are measured to obtain a measure of another attribute.

In software measurement there are some attributes which are measured directly, but typically the more sophisticated measures are obtained by indirect measurement.

Fenton [29] argues that although there is no universally agreed theory of measurement, most approaches are devoted to resolving issues such as what is and what is not measurement, which types of attributes can and cannot be measured and on what kind of scales, how we know if the attribute has really been measured, how the

measurement scales are defined, when an error margin is acceptable and when not, and which statements about measurement are meaningful and which are not. In Fenton's overview of the representational theory of measurement his discussion includes the empirical relation system, the representation condition, scale types and meaningfulness.

2.3.3.1 Empirical Relation System

He argues that direct measurement of an attribute must be preceded by intuitive understanding of that attribute. It is this intuitive understanding which leads to the identification of empirical relations between entities. The representation condition requires that when the empirical relation system is mapped to numerical relations, the empirical relations are preserved. The representation condition also asserts that the correspondence between empirical and numerical relations is two way. For example if the attribute 'height' of people with the relationship 'taller than' is mapped into a numerical representation, it does not matter whether it is mapped into centimetres or inches, the relationship will always hold.

Fenton [29] contends that, 'By having to identify empirical relations for an attribute in advance, the representational approach to measurement avoids the temptation to define a poorly understood, but intuitively recognisable, attribute in terms of some numerical assignment. This is one of the most common failings in software metrics work'. Classic examples are where attributes such as 'complexity' or 'quality' are equated with proposed numbers; for example, complexity with a 'measure' like McCabe's cyclomatic number [76], or Halstead's E [39], and 'quality' with Kafura and Henry's 'fan-in/fan-out equation' [42].

2.3.3.2 Scale Types and Meaningfulness

The scale type for an attribute is determined by the class of admissible transformations. For example where every admissible transformation is a scalar multiplication the scale type is ratio. The best known scale types in order of sophistication are nominal, ordinal, interval, ratio and absolute [90]. It is the formal definition of scale type based on admissible transformations which enables the rigorous determination of what kind of statements about measurement are meaningful.

'The notion of meaningfulness also enables us to determine what kind of operations we can perform on different measures. For example, it is meaningful to use means for computing the average of a set of data measured on a ratio scale, but not on an ordinal scale. Medians are meaningful for an ordinal scale, but not for a nominal scale. Again these basic observations have been ignored in many software measurement studies, where a common mistake is to use mean (rather than median) as measure of average for data which is only ordinal', notes Fenton [29].

2.3.3.3 Effort Estimation

In general cost modelling involves the prediction of time or effort for a software development project and typically has the form E = f(S). E is the development effort which may be expressed in person-months or development hours, and S a measure of system size. The function f typically involves other attributes such as complexity, required reliability, or programmer experience for example. Fenton's criticism of Boehm's COCOMO (COnstructive COst MOdel) model [12] to estimate development effort is that the unit of size is delivered source statements, which is an attribute of the final implemented system.

To use the estimation system at the specification stage requires the prediction of the product attribute, size, to enable it to be used as input into the model. This results in one difficult problem of predicting effort being replaced with the problem of predicting system size, which may be no easier. In Albrecht's [2] FPA this problem is avoided as the system size, in this case function points, is computed directly from the specification [29]. It is partly because of this characteristic that the function point metric was used as the basic size measure for this research project.

Software metrics should be validated on data other than the development data. Fenton [29] is of the opinion that the search for rigorous software measures has not been helped by the commonly held viewpoint that no measure is valid unless it is a good predictor of effort. He maintains that, 'An analogy would be to reject the usefulness of measuring a person's height on the grounds that it tells us nothing about a person's intelligence. The result is that potentially valid measures of important internal attributes become distorted. Consider for example, Albrecht's function points [2]. In this approach the unadjusted function count seems to be a reasonable measure of the

important attribute functionality in specification documents. However, the intention was to define a single measure as the main independent variable in prediction systems for effort. Because of this, a technical complexity factor is applied to unadjusted function points to arrive at the number of function points which is the model in the prediction system for effort. The technical complexity factor takes account of 14 product and process attributes in Albrecht's approach, and even more in Symon's approach [104]. This kind of assessment (to a measure of system functionality) is analogous to redefining measures of height of people in such a way that the measures correlate more closely with intelligence'.

In research done by Jeffery, Low, and Barnes [53] to compare various function point counting techniques they concluded that the technical complexity adjustment did not improve effort predictions, and that there is no significant difference between the unadjusted function points and function points in the development data they used. Similar research is also conducted in this thesis using artificial neural networks. This is done by comparing the prediction error of networks using function points as input to those using unadjusted function points. The hypothesis being assessed is whether the technical complexity adjustment contributes towards enhancing the unit of measure to facilitate more accurate effort estimates.

This author (Wittig) considers the example used by Fenton [29] (pg 205) in which he contends that trying to use function points to estimate development effort is analogous to using a person's height to predict his intelligence, as not particularly appropriate. It is considered that an analogy of predicting a person's weight using the person's height as independent variable would have been more appropriate. This is substantiated by an examination of some empirical data which suggests a reasonable correlation between unadjusted function points and development effort.

In section 2.4.2 where this author analysed development data it was found that the correlation coefficient between function points and effort for the Albrecht/Kemerer [31], Australian Software Metrics Association, and University of Natal datasets was typically in the range of about 0.60 to 0.80. It is intuitive to consider that there is some correlation between people's height and weight, but that typically one will find tall people who are not as heavy as some shorter people, and in an effort to be able to accurately predict a person's weight, other factors or attributes, such as a person's age,

genetic factors, eating habits etc should be included into the model. This example is thus considered a better analogy, as similarly in software development, once the system size is known or has been estimated in unadjusted function points, the influence of other factors which influence development effort should be incorporated into the prediction model to enable a more accurate estimate of development effort to be made.

2.3.4 Function Point Model Adequacy

In Chapter 5 several comparisons are made between the neural network models and some 'traditional' models. In this thesis neural network models are subjected to constant scrutiny. This section reviews the validity of the traditional models and their adherence to sound statistical techniques to enable a valid comparison to be made. Of necessity in some analyses the 'rules are broken'. For example some project data may just not have sufficient observations, and this section highlights the potential problems and places an interpretation of results in perspective.

2.3.4.1 Introduction

The function point count of a system is based on the conceptual rather than the physical model and represents the amount of function delivered to the user [2]. As a result the function count is not dependent on the development language or tools. FPA in its original form is designed to measure business-type applications and is not suited to measure scientific or real-time applications. Several adaptations have been proposed to accommodate the functionality provided by the algorithmic component of projects [56], as well as real-time and scientific systems [89].

FPA overcomes many of the criticisms associated with SLOC. For example, because of its language independence it is not subjected to the problem of accommodating various levels of languages [2]. Function points can be determined from the requirements specification, which is available relatively early in the development life-cycle [56]. Since function points are based on the user's external view of the system, non-technical users of the software system have a better understanding of what function points are measuring [60]. Even though function points appear to have advantages over SLOC as

a measure of software size for use in estimating development cost, there are few published cost estimation models that use it as the key input parameter [75].

Many of the models have been developed using empirical data for their calibration [16]. These models evaluate numerous factors across diverse programming environments. Three major problems have been identified with this approach [114].

- The inclusion of numerous independent variables frequently results in the problem that these are correlated [9]. Correlation among variables complicates both the statistical analysis and the interpretation of results. According to Vessey [114] a more effective approach would be to factor analyse the variables under consideration to determine the orthogonal factors underlying them, and to use those factors in further analysis. Alternatively if global variables are used this obviates the necessity for factor analysis.
- Empirical studies do not lend themselves to the control of extraneous variables. The approach is to collect a large volume of data and assume other variables not under study will be randomly distributed across the range of variables under investigation, so that there will be no biases or confounding results. Any variables which are not randomly distributed, should be included in the model.
- Many models have not been empirically tested or validated either within the same organisation or across organisations [114]. The ability of the models will be suspect and studies conducted indicate that the accuracy of these models is generally not very good [60].

These problems are evident also in function point analysis, and the following section discusses issues which relate specifically to the function point analysis models.

2.3.4.2 Regression analysis models

Estimation models are generally developed using regression analysis techniques. Various assumptions are made when using these models which generate a number of coefficients which need to be interpreted correctly. A major concern of the regression model is its adequacy or the explanatory power of the independent variable. This is typically measured by the coefficient of determination, R^2, and reflects the percentage of variability of the dependent variable explained by the independent variables [38].

However a large R^2 is not the only measure of a good model and in some regard may not even be the most important [75]. One of the fundamental assumptions of the regression model concerns the distribution of the residual or error terms. It is assumed that the residuals are distributed as independent, normal random variables with a mean of zero. To test this assumption the residuals are usually viewed on a scatter plot of the residuals versus the fitted values to confirm the independence and homoscedasticity, in addition to the normality plot for verifying the normality of the residuals. If these assumptions are seriously violated, a large R^2 may be of little importance. When one or more of the assumptions is violated, transformation of variables is often attempted as a remedy [75]. Unfortunately there is a temptation to conveniently ignore these violations if they can not be successfully remedied.

Typically datasets on which metrics research is conducted are not large (see Sections 2.4 and 2.5). A concern which cannot be ignored, especially if the number of observations is not large, is that of model stability. This refers the change in the model if any one of the observations is excluded from the analysis. Residual analysis is inadequate in determining model stability. This stability effect is summarised by Matson et al [75] as the ordinary least-squares criterion giving disproportionately large weights to cases which are extreme in the predictor variables in determining the fit, often resulting in small residuals for those extreme or high leverage cases. The most common method for assessing model stability is by case deletion, where each case is excluded from the data in turn and the coefficients are recalculated.

2.3.4.3 Albrecht and Gaffney Model

These analyses were conducted to illustrate the problems of model development and illustrates the complexity of this task. When the function point model was developed by Albrecht and Gaffney [2] they used a dataset of 24 projects, developed in three different languages. This is shown in Table 2.3. Of concern is the lack of proof that the various languages may be combined into a single homogeneous group.

Table 2.3 Albrecht/Gaffney Dataset

Project	Language	Function Points	KSLOC	Effort (Hours)
1	COBOL	1 750	130	102 400
2	COBOL	1 902	318	105 200
3	COBOL	428	20	11 100
4	PL/1	759	54	21 100
5	COBOL	431	62	28 800
6	COBOL	283	28	10 000
7	COBOL	205	35	8 000
8	COBOL	289	30	4 900
9	COBOL	680	48	12 900
10	COBOL	794	93	19 000
11	COBOL	512	57	10 800
12	COBOL	224	22	2 900
13	COBOL	417	24	7 500
14	PL/1	682	42	12 000
15	COBOL	209	40	4 100
16	COBOL	512	96	15 800
17	PL/1	606	40	18 300
18	COBOL	400	52	8 900
19	COBOL	1 235	94	38 100
20	PL/1	1 572	110	61 200
21	COBOL	500	15	3 600
22	DMS	694	24	11 800
23	DMS	199	3	500
24	COBOL	260	29	6 100

A closer examination shows that the system sizes ranged from 199 function points to 1902 function points, with only 4 systems being larger than 800 function points. The median is 506 function points while the mean is 648 function points.

To develop a model to estimate development effort from the function point count, ordinary least squares regression was used [2]. The fitted line for the dependent variable E (development effort), expressed as a function of the independent variable size FP, (function points) is as follows [75]:

$$E = -13.39 + 0.0545FP$$

A serious problem with this function is that for any systems smaller than approximately 246 function points the estimated effort is negative, which is unrealistic. On the other hand the explanatory power of the model is high with an $R^2 = 87.4\%$.

An examination conducted by Matson et al [75] of the residual plot suggests serious autocorrelation, while the normal probability plot suggests that the normality of the residuals is suspect. Also the influence plot shows that projects 1 and 2 (Table 2.3) are highly influential due in large part to their high leverage or extremeness in the independent variable (function points). The basic problem is that only 4 of the 24 projects are greater than 800 function points, and if these are removed a very different fitted line will result.

According to Matson et al [75] there are two possibilities to be considered here. The first is that the relationship is linear as Albrecht and Gaffney presumed, but the possibility exists that the error variance increases as project size increases, in which case the large function point values should be down-weighted in the regression. The other possibility is that the relationship between the independent and dependent variable is not linear, and an examination of the residual plot suggests this. The residual plot indicates that it may involve a quadratic term. In either event the model suggested by Albrecht and Gaffney appears to be inappropriate.

A data transformation was done by Matson et al [75] in which $\sqrt{E}$ (effort) is regressed against function points. The R^2 only improves marginally, but the residual plot now supports the independence and homoscedasticity of the error terms, and the normal probability plot is much improved. The other major benefit of this transformed model is that the prediction intervals are substantially narrower than those of the original model.

2.3.4.4 Kemerer Model

In an analysis of a cost estimation model which was developed by Kemerer [60] using 15 function point projects and linear regression, Matson et al [75] found that it is flawed similarly to the Albrecht model (Section 2.3.4.3). It has one project with very high leverage which influences the model disproportionately. If this case were not part of the dataset the derived model would be vastly different. With Kemerer's model the predicted values of the two smallest projects are negative. The model has a large mean square error which results in wide prediction intervals. As a result of this large mean square error, along with the violation of the regression model assumptions, it renders the resulting inferences virtually meaningless [75]. This would include the important

finding of the Kemerer analysis which had concluded that the Albrecht and Gaffney estimation model was validated by its independent study.

Matson et al [75] conclude that in addition to the observation of the assumptions of the ordinary least squares model, the reliance on the coefficient of determination or R^2 alone for assessing the appropriateness and strength of a regression model is erroneous. In addition to the R^2, the mean square error plays a vital role. For the development of a good regression model a fair amount of data as well as careful and thoughtful analysis of the model assumptions is required.

2.3.4.5 Log-linear Transformations

In another study on a dataset of 104 projects [75] all developed by one company which used experienced staff and encouraged the use of software development methodologies and tools, across a wide range of applications, the log-linear model, in which logarithmic transformations were applied to both the dependent and independent variables, resulted in an improved model, when compared to the normal linear regression model.

The stability of the model improved, in addition a visual analysis of the residual plots suggests improved results when compared to the original model. Standard tests [38] for linearity and the normality, independence, and homoscedasticity of the residuals confirmed the visual analysis. They concluded that the transformed model is superior to the original model in all aspects and in view of these results suggest that the log-linear relationship between effort and function points may apply generally to other studies.

2.3.4.6 Function Point Coefficients

The coefficients of the low, average, and high categories of the input, output, inquiry, internal file, and external interface file classifications were determined by 'debate and trial' [2]. These have been widely accepted in many thousand of projects world-wide without serious questioning [23] [56]. Matson et al [75] developed a regression model in which the independent variables were the constituent components (see Appendix A)

of function point analysis. In the original Albrecht model [1] there were only 4 classifications.

The initial Matson et al linear model resulted in a fit which exhibited many of the same shortcomings as the original Albrecht and Gaffney model described above. Another model was then developed in which the independent variables were subjected to non-linear transformations.

This resulted in a superior model in many respects. The residuals behaved better than in the original model, and the R^2 value was 97.5 percent. The mean square error for this model was 25.7, which is less than one quarter of that of the original Albrecht and Gaffney model, which resulted in prediction intervals less than half as wide. Of necessity the authors were restricted to the small original dataset. A larger dataset is essential before any model can be recommended with confidence.

The authors suggest that the key point is that improved results may be achieved by unbundling the function point variables into their constituent components. This direction requires further study but suggests strongly the possibility of more precise prediction in future studies.

2.3.4.7 Conclusion

Conte, Dunsmore, and Shen [16] suggest that a suitable measure of model error is the mean absolute relative error, and that this should be less than 0.25 for a good model. The original Matson et al model [75] had a mean absolute relative error of 0.87, while the transformed model, when converted back to its original form, gave a value of 0.71. Conte, Dunsmore, and Shen [16] also suggest that an acceptable prediction model will have at least 75 percent of all predictions within 25 percent of their actual observations. The Matson et al model [75] by their own admission was not within this criteria, with 68 percent of their estimates being within 25 percent of the actual effort. The complex and often non-linear relationships in the software development environment indicate that neural networks, which are known for their pattern recognition ability, warrant investigation to be used to capture these relationships in their weight space in an effort to develop improved models with reduced prediction errors.

2.3.5 Inter-Rater Inconsistency

2.3.5.1 Introduction

The inter-rater inconsistency describes the difference in a function point count which two individuals obtain measuring the same system. The presence of inter-rater inconsistency would introduce noise into project data. Function points have been criticised by its opponents as being, like lines of code, a controversial measure with uncertainty in counting definitions, in addition to its computation being based on subjective, rather than objective data [84].

In this study research results which report on the reliability and consistency of function point counts are reviewed.

2.3.5.2 Cost of Inaccurate Measurement

Planning and controlling software projects demand the capability to accurately and consistently measure the size of the software being delivered. To assess whether say a new tool and development methodology which promises to deliver a 20 percent improvement in productivity actually lives up to its claim, the system size must be measured reliably.

If this count is inconsistent, and on the one hand produces a count which is 20 percent below the true size of the systems being developed, it would indicate that the new tool and development methodology did not produce any improvement in productivity. As the new tool and implementation of a new methodology may be costly, it may be that the company decides against this, and incurs a large opportunity cost.

If on the other hand an assessment of the tool with the new methodology results in no development productivity improvement, and in this case an inconsistent count results in an exaggerated count of 20 percent, the company may decide to purchase the tool. Major development tools are usually costly, as is typically staff training for a new methodology. In this case the tool acquisition would have been a complete waste of expenditure and effort, with no benefits.

2.3.5.3 Function Point Acceptance

Function points have been widely accepted amongst practitioners and in the research community. Function points are being regarded by Dreger [23] as the best measure currently available, and it has been estimated that over 500 corporations worldwide are using function points [23]. One consulting company alone, Software Productivity Research Inc., has used function point counts for almost 6000 projects [54]. Function points have also been widely accepted by researchers and have been used for cost estimation [53] [60] [72] [78], software development productivity evaluation [11] [49] [57], software maintenance productivity evaluation [7], and software sizing [9].

2.3.5.4 Magnitude of Inter-Rater-Inconsistency–Rudolph Study

In a study done by Rudolph [91] in which 20 individuals independently determined the function point count of a system from the requirements specification, the values ranged within plus or minus 30 percent of the average count. This implies that if the average system size is 1000 function points, the lowest estimate would have been 700 function points, while the largest would have been 1300 function points. If such differences in function counts are typical of the current practice it would be cause for alarm as the largest count is not far from being double the lowest count. The cause for alarm would be due to the uncertainty that it generates as one does not know the variance of the current count from the 'true' count, which may not be the group average.

Rudolph considered that the main reason for such inconsistency to be the result of differing interpretation of the requirement specification. In his opinion function point counts which are done from the detailed design specification should not vary more than 10 percent above or 10 percent below the average count.

2.3.5.5 Magnitude of Inter-Rater Inconsistency–Low and Jeffery Study

In this study [72] two programs were counted by 22 experienced analysts from the program specification, which had been prepared by an experienced professional analyst/programmer according to commercial standards. The results are shown in Table 2.4

Table 2.4 Function Point Count Estimates

Program	Mean	Min	Max	Std Dev
1	57.7	26	159	26.3
2	39.6	15	76	13.4

The results reflect large inter-rater inconsistencies. It is noted from Table 2.4, for Program 1 the estimates ranged from minus 45 percent of the mean to plus 175 percent of the mean. For Program 2 the estimates ranged from minus 38 percent of the mean to plus 92 percent of the mean. These are wider margins than those reported by Rudolph [91] (Section 2.3.5.4). From another perspective, the highest function counts were approximately 5 times those of the lowest estimates. The reasons for such wide estimate ranges are discussed in Section 2.3.5.7.

These large variations were recorded when comparing inter-organisational counts. A review of the intra-organisational results reveals reduced variation and these are shown in Table 2.5 for those organisations which had used more than one rater.

Table 2.5 Intra-Organisational Comparison

Program	Organisation	Organisational mean	Minimum	Maximum
1	1	51.9	40	64
1	2	85.5	34	159
1	3	47.8	44	50
1	4	51.0	51	51
1	5	57.3	49	73
2	1	36.9	33	39
2	2	49.0	23	76
2	3	34.3	31	39
2	4	34.0	34	34
2	5	43.3	36	49

Despite the reduced variance the relatively large variation in organisational means and intra-organisational range still suggests that multi-rater means measures within some organisations may be unsatisfactory, as may be inter-organisational comparisons.

The size of programs which were used for the research were small. As is noted in Table 2.4, the mean count for Program 1 is only 57.7 function points, while that for

Program 2 is 39.6 function points. Typically many commercially developed systems are considerably larger than this [123] and it is uncertain what effect the added complexity of larger systems would have on the inter-rater inconsistency. On the other hand with small systems a single error in say not counting one internal file will reflect as a relatively large variation.

2.3.5.6 Magnitude of Inter-Rater Inconsistency–Kemerer Study

Kemerer [59] in his study to measure inter-rater inconsistency, assigned two function point counters from each participating organisation to measure the size of the projects of that organisation. For this study thus only the intra-organisational inter-rater inconsistency was assessed, in contrast to the Low and Jeffery study [72] (Section 2.3.5.5) where the function point counts of raters from various organisations were compared.

The raters for the Kemerer study were all experienced systems developers, having on average about 10 years experience. The average experience at measuring function points was about 1.5 years. In total 27 systems were measured, resulting in 54 observations. The average size of the systems was approximately 450 function points.

The inter-rater inconsistency is expressed as the percentage difference between the counts of each rater pair. The results show that the mean difference was 26.5 percent. This result is slightly distorted by the presence of a single outlier. The median difference value is 12.2 percent.

The results again suggest the presence of a significant problem with function point counting if accuracy is important. From such a research study it is not possible to assess to what extent the care taken by raters to ensure an accurate count under scrutiny represents the practical experience in the field. Intuitively there is also the suspicion that as a result of common influences within an organisation, that inter-rater inconsistencies across organisations would possibly be larger than the above results, as is suggested in the results given in Table 2.4 from the Low and Jeffery study [72]. If this is the case it makes accurate productivity comparisons between companies difficult.

2.3.5.7 *Causes of Inconsistency*

Rudolph [91] is of the opinion that the inter-rater inconsistency in his study resulted largely from differing interpretation of the requirements specification. For the Low and Jeffery [72] study possible explanations for the variation in estimates are given as follows:

- Inconsistency in assessing the complexity of the user function types. The user function types are classified as either simple, average, or complex, and inconsistency in their assessment results in different function point counts.
- Value judgements made when estimating the number of function points. For example is the generation of two reports which differ only in the order in which the line items are listed counted as one or two outputs? It may be an inherent feature of function point analysis that the count is sensitive to such factors and that clear definitions are essential for all the numerous possible situations. The result of this is the requirement of a voluminous counting manual, such as the IFPUG Counting Practices Manual Release 4.0 [35]. An accurate measurement necessitates thorough knowledge of its contents restricting the number of available raters, or alternatively it will result in a slow and costly count while continually having to refer to the manual.
- Different interpretations of the specification. Again only clear definition of all those aspects which may cause ambiguity will resolve the issue.
- Perception of object boundaries. The question here is whether an internal file belongs to the program being assessed or to the overall system, in which case the file is treated as an external interface file.
- The objectives of the function point counter. Low and Jeffery [72] argue that since function points are used by some organisations to measure staff productivity, there may be a subconscious tendency to adjust the count based on the perceived program difficulty. If this is the case that it significantly influences the function point count as the authors suggest, it appears to substantiate the suggestion that function point counting is not sufficiently objective [84].

- Analyst experience in *a priori* counting. While in this study all raters were experienced in function point counting, some of them only had *a posteriori* experience.

In the Kemerer and Porter study [61] on function point reliability 45 respondents, all IFPUG members, returned completed questionnaires which were analysed to identify 11 probable areas of variation for further investigation. This was conducted at three case study sites, with the following topics indicating sources of greatest variability:

- The counting of backup files was identified as an area of consistent variation and this resulted in a variation ranging from 17 to 31 percent. This item was the largest single source of variation.

- Counting menus was identified as a likely source of variation and the case study revealed variations ranging from 3 to 11 percent. This source was identified as worthy of further study. When this study was conducted the then current IFPUG Counting Practices Manual (CPM) was version 3.0, which did not count menus at all. A concern expressed by the authors is that as Graphical User Interfaces (GUIs) become more widespread, and menuing becomes the rule rather than the exception, this issue will become more significant in terms of their impact on the reliability of function point counts. In Release 4.0 the counting practice is now to count GUI 'pull-down' menus and list boxes [35].

- External interface file transaction counting is another area identified as a source of variability. The case study revealed a 16 percent variability in the one site where this was applicable.

- Counting of add/change/delete transactions resulted in a variance of 6 percent where this was counted as a single output. At the other two sites the counting convention followed the CPM instruction. If these sites had followed the other site's counting convention, the variability would have been 10 and 25 percent respectively.

- Counting help screens was applicable to only one of the case sites, and this resulted in a 6 percent change in the overall count. With the demand for internally developed software to increasingly match the functionality of off-the-shelf software, which is typically extensively equipped with 'Help' and other facilities, it is reasonable to

expect these functions to account for more of the overall functionality than at present, and this may result in greater count variations.

The authors also conducted a worst case scenario analysis, assuming that a site was unfortunate enough to have chosen every variant to maximise the difference between its count and the CPM standards count. The overall difference is not just simply the sum of all the variants, as not all of them are independent. To avoid logically inconsistent counting, some mutually exclusive variants were excluded. The worst case resulted in a variation of 43 percent, and the authors comment that this is not inconsistent with previous research results [59].

2.3.5.8 Conclusion

The perception of function points being unreliable has undoubtedly slowed their acceptance as a metric [61]. To reduce the effect of inter-rater inconsistency and ensure measurement reliability an organisation could use only a single individual to conduct all their function point counts. This though reduces flexibility and makes inter-organisational comparisons difficult, as well as intra-organisational comparisons when the regular rater leaves the company. Another alternative suggested is that multiple raters should be used for all systems, and their counts then averaged. Function point counting is a tedious and costly procedure. For the Kemerer study [75] this averaged 1 work hour per 100 function points counted. This cost would mitigate against multiple counts.

To reduce human error and the cost of function point counting, a number of proposals have been made to automate function point counting or a variant of function point counting [8] [111] [113] [121]. Generally these are still in the conceptual stage and have not been validated on any software projects. Rask [87] has developed rules for the transformation between Structured Analysis (SA), Jackson's System Development (JSD) and unadjusted function points. In experiments these algorithmic transformations were verified to have preserved the unadjusted function point count and this allows the function point count to be determined automatically from CASE tools which employ these design methodologies.

Another critical precursor to the widespread acceptance of an automatic function point counting facility is a clear set of well-defined measurement conventions [61], without which measurement inconsistencies will not be eliminated.

Precautions also need to be taken when measuring software development effort, as it forms part of the productivity equation and inconsistencies here could result in erroneous results. A clear definition is required of which development functions form part of the total development effort. For example are staff, or end-user training included? In some countries, notably the United States, most software professionals are exempt from overtime payments, and for this reason it is not recorded on any time-sheets. Consequently it is often not included in the total development hours of a project. According to Jones [56] this may cause a discrepancy which can be between 20 and 30 percent.

Accurate software size measures are essential precursors to the development and calibration of accurate estimation and productivity analysis models. These models can be no more accurate than the underlying size measures which are used as inputs to the models.

2.4 Productivity Variations in Research Data

2.4.1 Introduction

Planning and control systems are based on performance indicators as determined by production output. This applies also to the software development industry. To have confidence in software development productivity measurements, these need to be consistent, and there has to be explanation or at least acceptance of any productivity variation.

To estimate development effort the system size is multiplied by a 'delivery rate' function, which is typically the inverse of the development productivity function. A trivial illustration is for example if a system is 1,000 FPs and productivity is 0.1 FPs/hour, then system development effort will be 10,000 hours. The crucial link between system size and development effort is thus development productivity. To

establish what the appropriate development productivity is, it is necessary to analyse past developments. If an empirical analysis reveals large variations in productivity, it becomes necessary to understand why these occur, so that these circumstances and their influence can be incorporated into the productivity function which is used to estimate development effort.

2.4.1.1 Technical Complexity Adjustment

If all the variations in development effort are explained by the technical complexity adjustment (TCA) factors of FPA, then a common productivity function could be applied to all developments to obtain an accurate estimate of effort. The TCA allows for adjustment of +35% to -35% of the unadjusted function point value. This implies that FPA could accommodate variations of this magnitude and no further adjustment would be required.

This author conducted a study of datasets of software developments to analyse the productivity variations. Large productivity variations imply that the causes of these cost drivers have to be identified and quantified so that they can be incorporated into the productivity function for accurate effort estimation [123].

2.4.1.2 Reasons for Productivity Variations

There are numerous reasons for productivity variations. One reason is inter-rater inconsistency in the size determination (see section 2.3.5). Another reason is that an inappropriate measure of software size which does not accurately reflect the true size will cause variations (see section 2.3.4). A function point represents the amount of functionality which a system provides to the user. This is an arbitrary measure, derived by debate and trial, and the weight coefficients were established to reflect the differences in functionality for the various system classifications. Thus if these or the technical adjustment size drivers which are used to determine the function point count are not appropriate, it may result in a size measure which is inconsistent across similarly sized systems. This implies that if two similar systems are developed with the same amount of effort, the productivity would differ, only because of the size discrepancy. Such a result would lower confidence in the productivity measure.

Inherent in software developments are various factors, called cost drivers, which affect the amount of effort that was required to develop a system. The productivity variation could be the result of numerous cost drivers which result in one system being produced with less effort than another. The level of programmer capability and requirements specification volatility are examples of cost drivers.

Large differences in productivity due to an inappropriate size measure or large inter-rater inconsistencies are not acceptable. If the variation due to the cost drivers is not explicable or credible, it will reduce confidence in the performance measure. There has not been agreement in the identification of significant cost drivers, nor their effect on productivity.

The study conducted by this author [123] investigated productivity variations that occur in software developments. If it is assumed that software size models are credible and not based on poor fundamental concepts and calibration, and that variations in productivity are also not largely due to inter-rater inconsistencies, then the variation must be explained by the other cost drivers. The objective of this study was to determine the magnitude of the cost driver adjustment which is necessary for the productivity measurement to be consistent and useful to management. This implies that the cost driver size adjustment has to be in direct proportion to the productivity variation.

2.4.2 Productivity Analysis

Three sets of empirical data were analysed to examine the variations in productivity which are inherent in software development [123]. The basic size (output) unit is function points. Productivity is described as output per unit of input, and is expressed in either function points developed per development hour or per person month.

2.4.2.1 Albrecht/Kemerer Data Set

The first 22 entries of this dataset are from a database developed by Albrecht, and the last 14 were from Kemerer's dataset [60] which he used for his empirical validation of software cost estimation models [31]. The 36 projects are shown in Table 2.6. The Unadjusted Function Point (UFP) count, the Function Point (FP) count, and the

Thousand Source Lines of Code (KSLOC) are shown. Of the 36 projects, 30 (83.3%) were developed in COBOL. Four of the other projects were developed in PL/1. If all projects not written in COBOL are excluded from the data set the analysis results do not change substantially, and they have thus not been excluded. The unit of effort is Person-Months (PM).

Table 2.7 shows the different rates of productivity. The large variations in all three size measures is noted. For function points this varies from 2.08 to 13.73 FPs per PM. This shows that in project 34 the productivity was about 6.6 times higher than in project 27. This is a very large difference which has to be explained by the cost drivers. Using SLOC as size measure the productivity range is even larger.

Also included in the table is a comparison of the number of SLOC per FP. Again the differences are very large in this mainly COBOL data set. This implies that without any cost driver adjustment, a consistent SLOC per FP conversion will have a high variance.

Some researchers claim that system size is an important cost driver [49] and the effect of cost drivers is reflected in many effort estimation models [16]. Figure 2.1 depicts a graphical comparison of productivity and system size. A visual inspection does not indicate a clear relationship between system size alone and productivity. The correlation coefficient between system size and productivity is only 0.222. This does not imply that system size is not a significant cost driver. The complex and often non-linear interrelationships between the cost drivers make it difficult to establish their relationships and effects. Techniques such as the use of artificial neural network models may be useful to establish these.

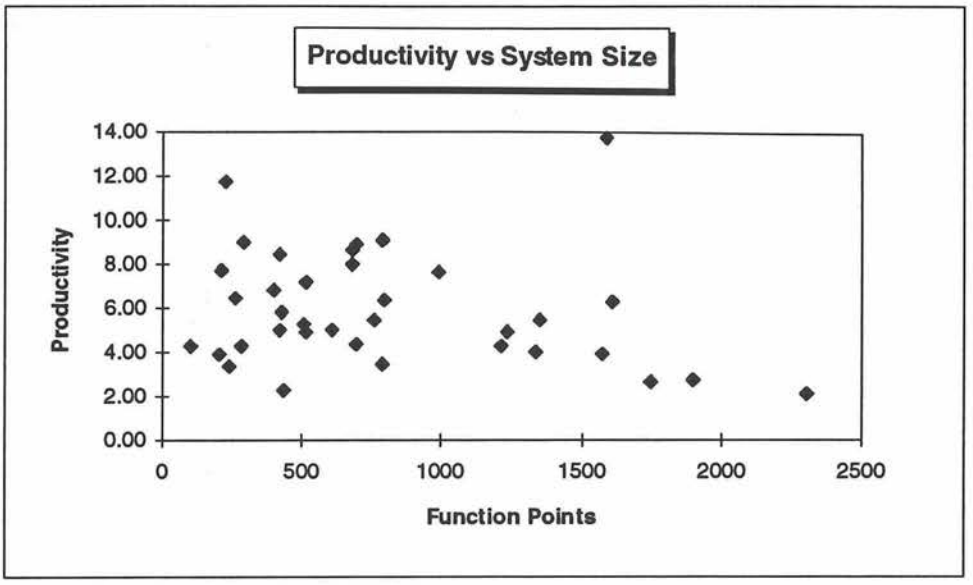


Figure 2.1 Productivity and System Size Comparison

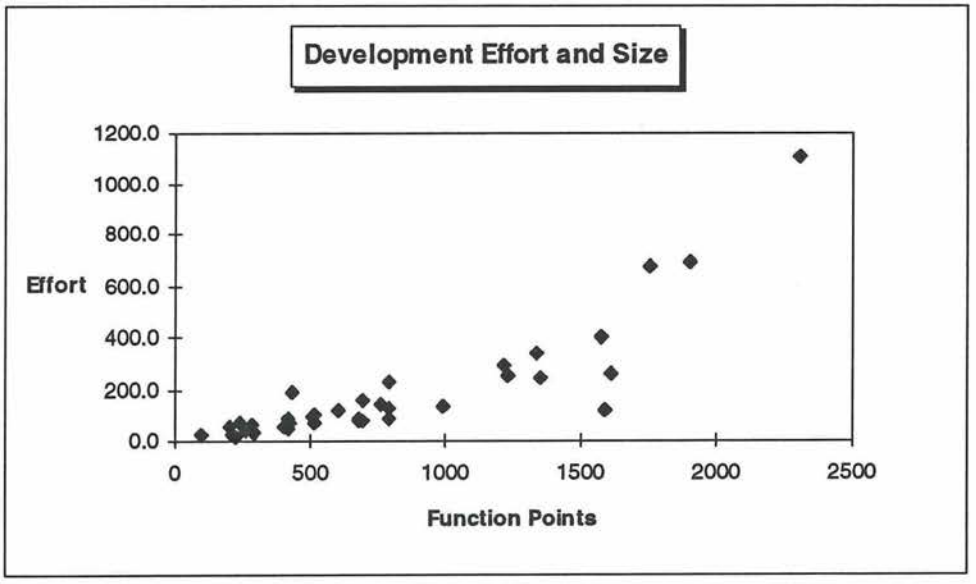


Figure 2.2 Development Effort and System Size

Figure 2.2 shows a scatter graph comparing system size and development effort. The development effort measured in Person Months (PMs) and is reflected on the Y-axis, while system size measured in function points is shown on the X-axis. As expected there appears to be a trend of increased effort as size increases, with a correlation coefficient of 0.852. Caution should be exercised in not considering software size in isolation because of the complex interrelationships with the other cost drivers.

Table 2.6 Albrecht and Kemerer Data Set

Project	KSLOC	UFP	FP	Effort
1	130	1750	1750	673.7
2	318	1902	1902	692.1
3	20	522	428	73
4	54	660	759	138.8
5	62	479	431	189.5
6	28	377	283	65.8
7	35	256	205	52.6
8	30	263	289	32.2
9	48	716	680	84.9
10	93	690	794	125
11	57	465	512	71.1
12	22	299	224	19.1
13	24	491	417	49.3
14	42	802	682	78.9
15	40	220	209	27
16	96	488	512	103.9
17	40	551	606	120.4
18	52	364	400	58.6
19	94	1074	1235	250.7
20	110	1310	1572	402.6
21	24	694	694	77.6
22	29	263	260	40.1
23	254	1010	1217	287
24	214	881	788	86.9
25	254	1603	1611	258.7
26	41	457	507	95.5
27	450	2284	2307	1107.3
28	450	1583	1338	336.3
29	50	411	421	84
30	43	97	100	23.2
31	200	998	993	130.3
32	39	250	240	72
33	129	724	789	230.7
34	289	1554	1593	116
35	161	705	691	157
36	165	1375	1348	246.9
Mean	116	794	800	185.0
Std Dev	117	545	560	224.0
Min	20	97	100	19.1
Max	450	2284	2307	1107.3

Table 2.7 Project Productivity

Project	UFP/PM	FP/PM	KSLOC/PM	KSLOC/FP
1	2.60	2.60	0.19	74.29
2	2.75	2.75	0.46	167.19
3	7.15	5.86	0.27	46.73
4	4.76	5.47	0.39	71.15
5	2.53	2.27	0.33	143.85
6	5.73	4.30	0.43	98.94
7	4.87	3.90	0.67	170.73
8	8.17	8.98	0.93	103.81
9	8.43	8.01	0.57	70.59
10	5.52	6.35	0.74	117.13
11	6.54	7.20	0.80	111.33
12	15.65	11.73	1.15	98.21
13	9.96	8.46	0.49	57.55
14	10.16	8.64	0.53	61.58
15	8.15	7.74	1.48	191.39
16	4.70	4.93	0.92	187.50
17	4.58	5.03	0.33	66.01
18	6.21	6.83	0.89	130.00
19	4.28	4.93	0.37	76.11
20	3.25	3.90	0.27	69.97
21	8.94	8.94	0.31	34.58
22	6.56	6.48	0.72	111.54
23	3.52	4.24	0.89	208.71
24	10.14	9.07	2.46	271.57
25	6.20	6.23	0.98	157.67
26	4.79	5.31	0.43	80.87
27	2.06	2.08	0.41	195.06
28	4.71	3.98	1.34	336.32
29	4.89	5.01	0.60	118.76
30	4.18	4.31	1.85	430.00
31	7.66	7.62	1.53	201.41
32	3.47	3.33	0.54	162.50
33	3.14	3.42	0.56	163.50
34	13.40	13.73	2.49	181.42
35	4.49	4.40	1.03	233.00
36	5.57	5.46	0.67	122.40
Min	2.06	2.08	0.19	34.58
Max	15.65	13.73	2.49	430.00
Mean	6.10	5.93	0.81	142.32
Median	5.21	5.38	0.63	120.58

2.4.2.2 ASMA (Victoria) Dataset

By June 1993 the Australian Software Metrics Association (Victoria branch) had collected 56 sets of function point project data [6]. Of this set there were 12 new 4GL (Fourth Generation language) projects and these are shown in Table 2.8. The Project ID is the identification number as given in the ASMA project report. The size range is large, varying from 220 to a very large project of 5684 function points. Productivity is measured as function points developed per hour. As in the previous data set (Section 2.4.2.1) the productivity range is very large. The lowest productivity is 0.072 function points per hour, while the highest is 0.833 function points per hour, which is over 10 times higher.

Again, unless it is assumed that these differences are mainly due to development team capabilities, the other cost drivers are important in reconciling these, if the productivity measurement is to be meaningful. Figures 2.3 and 2.4 show graphically the system size, and productivity and effort respectively. Again the correlation coefficient between system size and productivity is low (0.226) and between system size and development effort is relatively high (0.803).

Table 2.8 ASMA (Victoria) 4GL Data

Project ID	FP	Hrs	FP/Hr
1	615	3567	0.172
6	4562	26003	0.175
10	274	521	0.526
11	220	264	0.833
20	1457	14716	0.099
22	3180	32754	0.097
24	597	2507	0.238
42	365	4307	0.085
43	586	4922	0.119
48	1134	15649	0.072
49	1362	3133	0.435
56	5684	20462	0.278
Min	220	264	0.072
Max	5684	32754	0.833
Mean	1670	10734	0.261
Median	875	4615	0.174

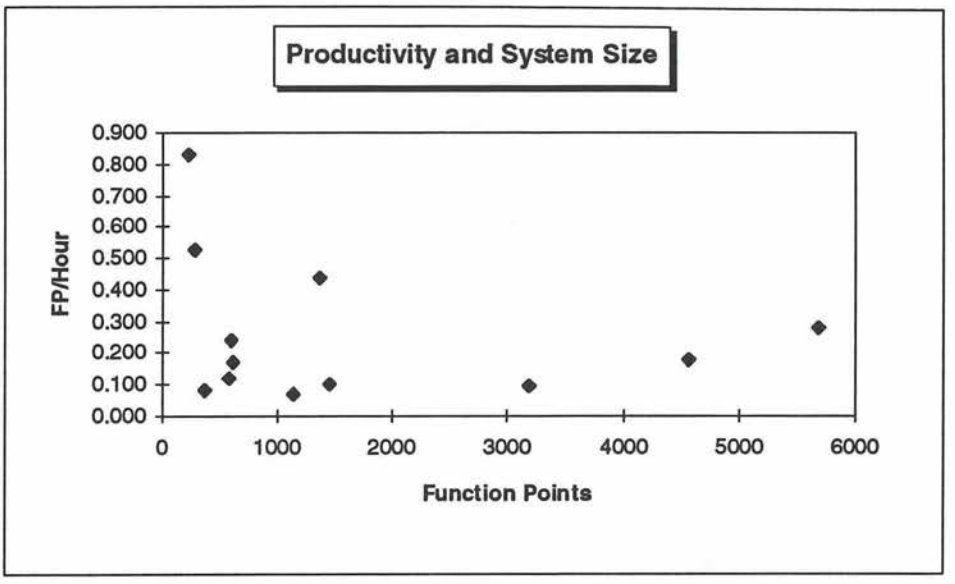


Figure 2.3 ASMA Productivity Data

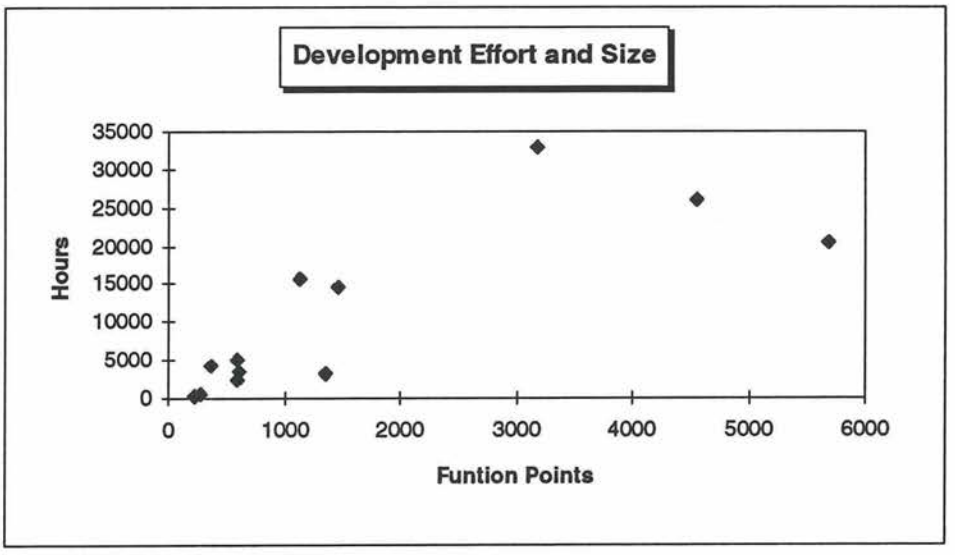


Figure 2.4 ASMA Development Effort Data

Of interest in the ASMA dataset, which also included 25 projects developed in the 3GL environment, is the productivity comparison between the 3GL and the 4GL projects. The 3GL data of new development projects is shown in Table 2.9. It is noted that the range in productivity is even larger than for the 4GLs. A comparison of the average development productivity between 3GLs and 4GLs is shown in Table 2.10.

Despite the maximum productivity being the same in both cases, the mean productivity of the 4GL environments was almost 80 percent higher than that of the 3GLs.

Table 2.9 ASMA (Victoria) 3GL Data

Project ID	FP	Hrs	FP/HR
3	82	541	0.152
4	502	9287	0.054
5	846	16158	0.052
7	309	2596	0.119
8	1223	66531	0.018
13	578	5086	0.114
14	86	413	0.208
15	171	205	0.833
16	151	1027	0.147
17	125	513	0.244
18	203	2416	0.084
19	1093	2514	0.435
21	3332	48980	0.068
31	77	1093	0.070
33	244	3001	0.081
34	144	4378	0.033
35	171	2839	0.060
36	13	1014	0.013
37	126	1033	0.122
44	1186	3677	0.323
46	4411	59549	0.074
47	465	10184	0.046
51	231	4112	0.056
53	2944	21491	0.137
54	211	1857	0.114
Min	13	205	0.013
Max	4411	66531	0.833
Mean	757	10820	0.146
Median	231	2839	0.084

Table 2.10 Comparison of 3GL and 4GL Productivity

	Number of Projects	Productivity (FP/Hr)	Min	Max
3GL	33	0.146	0.013	0.833
4GL	12	0.261	0.072	0.833

2.4.2.3 *University of Natal Data Set*

Data for this set was gathered from 15 commercial 4GL software developments [119], across a large range of sizes, as is shown in Table 2.11. System size is measured in unadjusted function points, while productivity is measured in unadjusted function points per development hour. As in the other two data sets the productivity range is very large. The lowest productivity is 0.05 function points per hour, while the highest is 2.14 function points per hour.

Table 2.11 University of Natal Data Set

Project No	UFP	Hrs	UFP/Hr
1	1842	5027	0.37
2	905	1680	0.54
3	4191	13300	0.32
4	208	98	2.12
5	342	588	0.58
6	164	450	0.36
7	29	40	0.73
8	4113	81270	0.05
9	3486	35000	0.10
10	286	240	1.19
11	214	100	2.14
12	2758	4864	0.57
13	1913	1200	1.59
14	4669	2500	1.87
15	850	1400	0.61
Min	29	40	0.05
Max	4669	81270	2.14
Mean	1731	9850	0.88
Median	905	1400	0.58

For these productivity measures to be useful and have credibility, the cost driver coefficients which explain these differences need to be established. Figures 2.5 and 2.6 show graphically system size in terms of productivity and development effort respectively. As on the previous two occasions the correlation between system size and

productivity is low (0.254), while the correlation between system size and development effort is higher at 0.575.

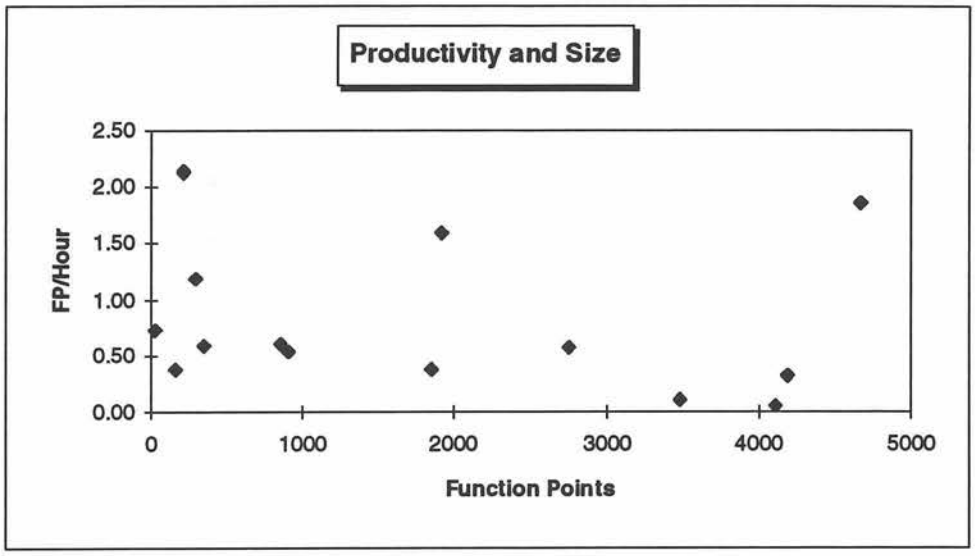


Figure 2.5 System Size and Productivity

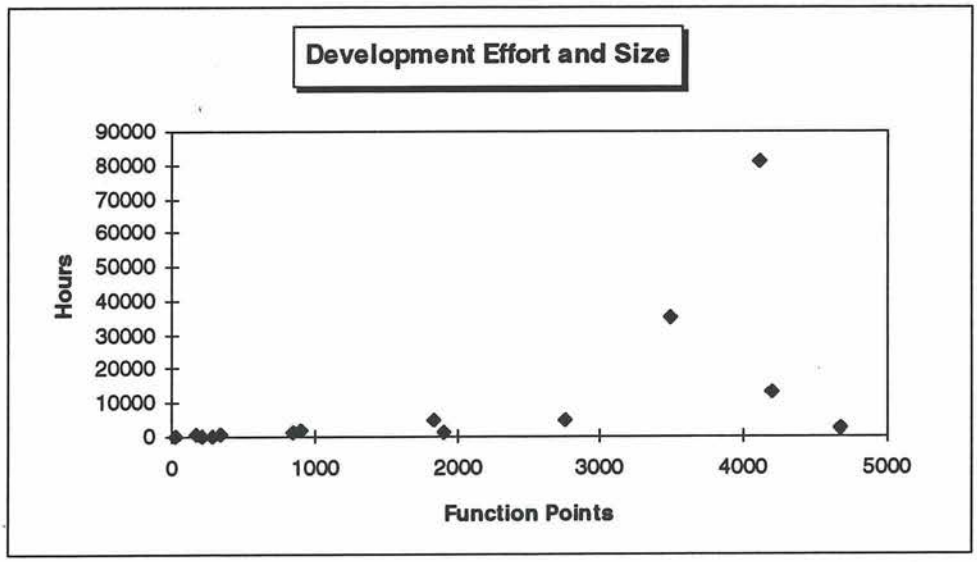


Figure 2.6 System Size and Effort

2.4.3 Conclusion

The analysis of these empirical data sets revealed large differences in productivity which are inherent in software developments. For productivity measures to be useful it is necessary to identify and quantify the significant cost drivers. Graphical inspections do not readily reveal any clear relationship between system size and productivity. Other significant cost drivers also have to be considered. Their inter-relationships are complex and often non-linear, and are difficult to establish. A pilot study conducted by the author [120] has indicated that artificial neural networks may be useful in providing a better understanding of some of these factors.

Multiple regression and other statistical techniques have traditionally been used to capture these relationships, and these are reflected in numerous effort estimation models with various coefficients which have been developed [16]. Research appears to indicate that it is difficult to generalise the effect of cost drivers across different development environments, and models may have to be calibrated within organisations or homogeneous developments [40] [50] [66].

Artificial neural networks are recognised for their ability to provide good results when dealing with problems where there are complex relationships between inputs and outputs, and where the input data is distorted by high noise levels [92]. The software development environment from which development effort estimates are generated, are characterised by these attributes. Banker et al [7] suggest that linear models are likely to be inadequate representations of the development process. Artificial neural networks are universal approximators and thus have the ability to model non-linear relationships. This implies that any lack of success in modelling these must arise from either inadequate learning, insufficient numbers of hidden units or a lack of a deterministic relationship between input and target [46].

2.5 The Effect of Development Attributes

2.5.1 Introduction

The claims of consistent and accurate software cost estimation made by some organisations is not supported by results of published empirical research [109]. The performance of cost estimation models is discussed in section 2.6 and the results are disappointing. An analysis of empirical data reveals large differences in software development productivity. These differences, which typically are caused by development effort attributes or cost drivers, have to be explained or modelled to facilitate accurate effort estimates [109]

Unfortunately there is no unanimity in the research community on which development attributes affect effort and productivity, and to what extent [16]. Published studies appear to reveal contradictory and inconclusive results [9] [16] [49] [51] [52] [64] [68] [114] [119]. This lack of agreement may in part be responsible for the lack of uniform project data being recorded. This lack of data inhibits the research effort into modelling the cost drivers.

Many studies note the need for calibration of estimation models to the development environment [12]. The lack of agreement on which factors in the environment are important cost drivers [16] [56] [60] [114] makes their identification difficult. COCOMO [12] and numerous other models which were developed in the third generation (3GL) environment predict a power functional relationship in terms of development effort.

The increasing non-linear growth in estimated development effort as size increases implies that there are *reducing* returns to scale. Other models, for example the Walston-Felix model [16] [115] reflect *increasing* returns to scale, and predict that development effort will increase proportionately less than the increased size of the system. Albrecht and Gaffney's function point analysis model [2] assumes a *linear* increase in development effort as the size of systems increases. Jeffery on the other hand [49] states that initially increasing returns to scale can be expected as size increases but that beyond a certain point decreasing returns to scale can be expected.

There is also no general agreement on the magnitude and nature of these and other cost driver coefficients. Albrecht's model [2] has 14 complexity adjustment factors which are presumed to be independent and additive. Other function point models have additional adjustment factors [56] [89] [103].

Jeffery and Low's [50] findings do not support the use of uncalibrated generic effort estimation tools. Kitchenham [63] states that results obtained from the analysis of the MERMAID 2 dataset suggest that relatively few cost drivers are important in any particular environment and that these are likely to differ from one environment to another. In this study Kitchenham also concluded that the technical complexity adjustment factors are not independent, as is presumed in function point analysis.

2.5.2 Finnie and Wittig Study

Here the effect of system and team size on fourth generation language development productivity were studied [32]. Fifteen commercially developed projects were analysed and it was found that productivity decreased as the development team size was increased. Intuitively it would be expected that the larger the system being developed, the larger the development team. The Spearman rank correlation coefficient of 0.74 of system size and development team size appears to substantiate this. This confuses the analysis as to whether increased team size or system size contributes to the reduction in productivity.

Numerous models were developed and tested, of which the semi-log model fitted the data best. The independent variables were the average team size, the average number of function points developed per team member, and the natural log of the system size. These were regressed against the natural log of development productivity. The model yielded an adjusted coefficient of determination (adj R^2) of 0.72, and the coefficients of the independent variables were significant at 0.05.

This model exhibits some interesting characteristics, and is graphically depicted in Figure 2.7, depicting three system sizes of 100, 200, and 400 function points. On the smallest system as the team size is increased, the productivity decreases, reflecting the effect of this cost driver. A similar effect is detected in the 200 function point system. In the 400 function point system, initially as the team size is increased from one to two

member, there is an increase in productivity, after which any increase in team size reduces productivity and increases total development cost.

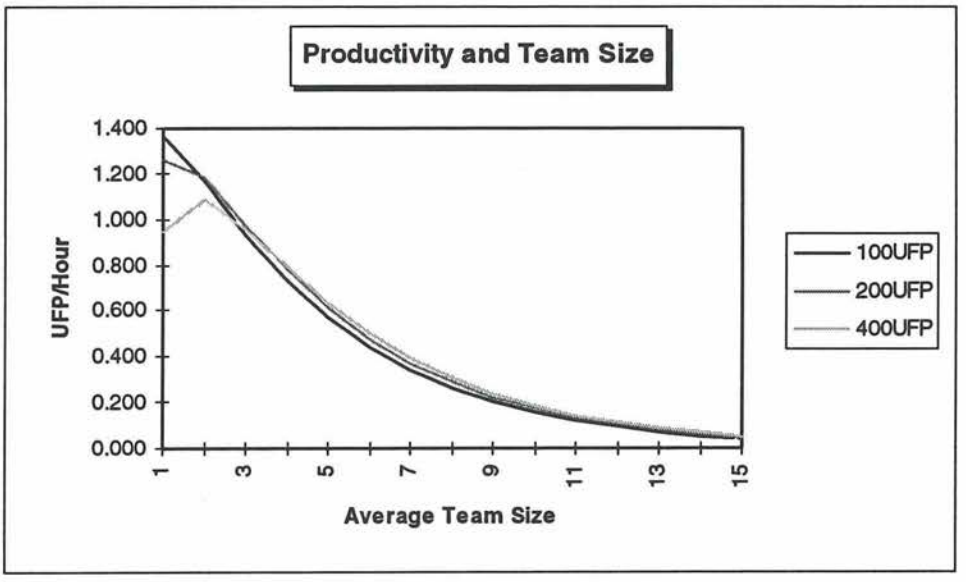


Figure 2.7 Productivity vs Team Size vs System Size

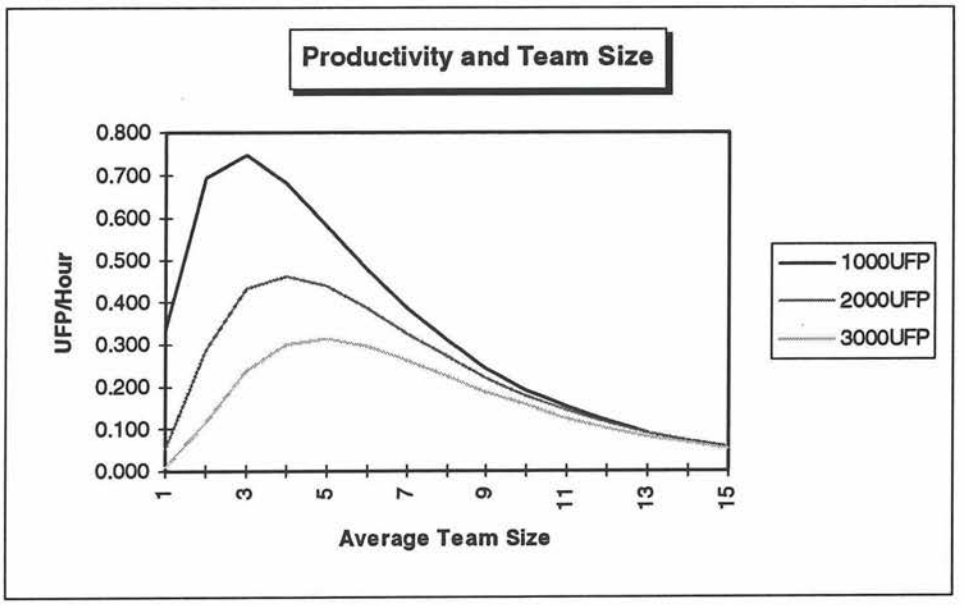


Figure 2.8 Productivity vs Team Size vs System Size

In Figure 2.8 the same model is depicted with systems of 1000, 2000, and 3000 function points. To note here is that as system size increases the development productivity appears to decrease, as can be seen by a general comparison of the three curves. Again in each case there is an initial increase in productivity, before a further increase in team size has a depressing effect on productivity.

Of interest is that this model indicates that each system size has an optimum team size to minimise development costs, at which stage development productivity will be maximised. The optimum team size indicated by this model appears to increase with system size. This trend of predicting larger optimum team sizes as system size increases supports the intuitive perception that larger systems are better able to employ the larger development teams.

A possible explanation of this may be due to initial specialisation of tasks which result in productivity gains, after which the ever increasing communication and control overhead of managing larger teams counteracts this effect to eventually start reducing productivity.

The above model facilitates an examination of the trade-offs which are considered for a project development. From the model it is relatively easy to derive the predicted total development hours as well as the predicted development schedule or elapsed time which will be required to develop the system. To develop a project at the lowest cost, the total development hours must be minimised, and this occurs when productivity is at its peak.

In the example of the 1000 function point system used above, increasing the team size beyond the initial optimum of three will result in reduced predicted productivity. Initially though the proportionate increase in team size is larger than the proportionate decrease in productivity, which will result in a reduction of time to develop the system. Further increases in team size eventually lead to reductions in productivity which are proportionately larger than the proportionate increase in team size, and from this point on any further increases in team size will result in a predicted increase of development schedule. This is shown graphically in Figure 2.9.

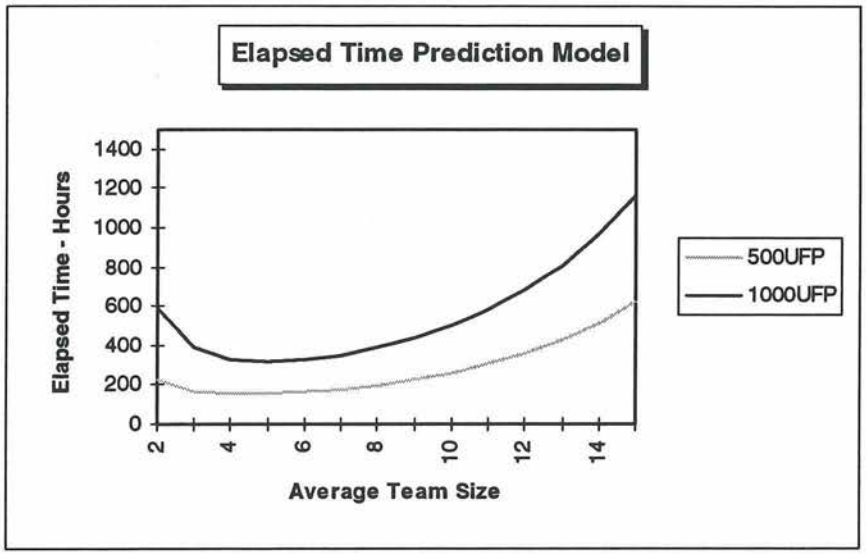


Figure 2.9 Predicted Elapsed Time vs Team Size

The main problem with such a model is that it is developed from only 15 observations. It is inconceivable that the many complex interrelationships can be adequately modelled from so few observations. Matson, Barrett and Mellichamp [75] are critical of models derived from limited data sets, and point to the problem of model instability and the influence of high leverage observations having an inordinate influence on the model coefficients. They also warn of the danger of using only the coefficient of determination for model selection.

2.5.3 Jeffery Study

Jeffery [49] studied the relationship between team size, experience, and attitudes. Of particular interest were the productivity implications. The study consisted of three sets of trials. The first was to analyse the impact of elapsed time compression on productivity and thus on total development effort. Data was collected on 47 projects which were developed by four organisations. Jeffery noted that, 'The data in this study did not support a strong relationship between productivity and elapsed time variation above or below what would normally be expected for the project's size. Specifically,

there was no support for the notion that elapsed time compression necessarily results in decreased productivity in the environment studied.'

These findings were contrary to those of Boehm [12] and Putnam [85]. Jeffery had used the variable actual elapsed time divided by the estimated elapsed time as a measure of elapsed time compression, and it was suggested by Jeffery that a better surrogate for project stress might be the maximum staffing level used on a project of a given size. For example, if the maximum staffing level used on a project was greater than normal then this would indicate project stress or completion pressure.

This raises another issue, which is how to represent the development team size as input to a model. Finnie and Wittig [32] used the average team size, whereas Jeffery in this study uses the maximum staffing level. Neither are totally satisfactory. On the one hand unusual circumstances may for a very short time have resulted in a large maximum staffing level, which for that short period may only have had a minimal effect on productivity.

On the other hand, during the analysis phase staffing levels are typically lower than during the design phase. Taking an average staffing level ignores the possible cost driver effect which project pressure may have. For example, a project which is understaffed initially and then requires severe staff loading due to project pressure to complete the project within the time specified, may have the same average team size as one which is staffed at the appropriate levels throughout. It appears as if both the average, as well as the staff level distribution are important and should be incorporated into a model.

In the second set of trials Jeffery [49] used system size and the maximum staff level as the independent variables in a stepwise log transformed regression model. This resulted in the following model form:

$$\text{Productivity} = c \times \text{SIZE}^{x} \times \text{MaxStaff}^{y}$$

Three models were developed, one for each of the development languages, namely COBOL, Basic, and Focus. In all cases the x coefficient was $0 < x < 1$, and the y coefficient was < 0. Jeffery concludes from this that for the projects used in the study, development productivity will increase with size, and decrease with staff levels. Unfortunately the data sets used for model calibration were small, and the model risks

highlighted by Matson, Barrett, and Mellichamp [75] and mentioned in Section 2.3.4 are applicable here as well.

The following analysis conducted by this author illustrates the point. Five of the models, for a maximum staffing level of five, for different languages are illustrated graphically in Figure 2.10, using the equation from Jeffery [49].

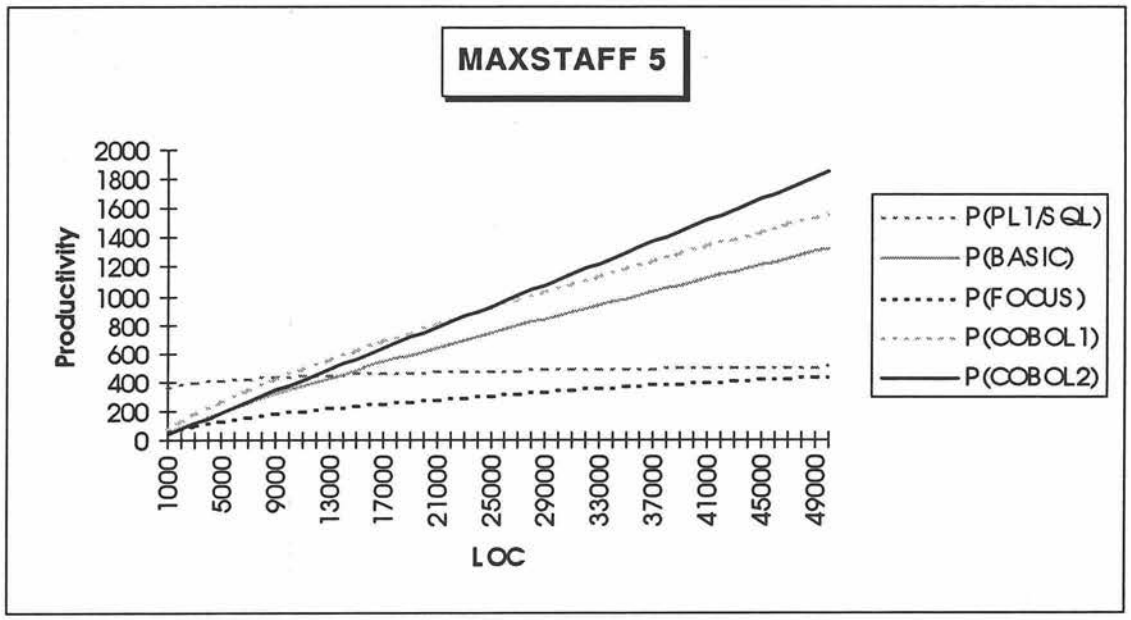


Figure 2.10 Productivity vs Size for MaxStaff of Five

In this model as the system size, Lines of Code (LOC), increases the productivity increases. The differences between the various development languages is notable. This could be partly due to differences in staff experience on the various sites. On closer observation the COBOL 2 model appears to have an anomaly. The increase in productivity is almost equal to the increase in project size. This is shown in Table 2.12. A result of this is that there is hardly any increase in predicted development effort, implying the cost of developing a 1,000 lines of code project is virtually the same as a 50,000 lines of code project, which practical experience and intuition tends to contradict. A larger data set may have eliminated such anomalies.

Table 2.12 COBOL 2 Model

LOC	Productivity	Dev Effort
1000	38	26.0
5000	189	26.4
10000	376	26.6
15000	561	26.7
20000	746	26.8
25000	930	26.9
30000	1115	26.9
35000	1298	27.0
40000	1482	27.0
45000	1665	27.0
50000	1848	27.1

Jeffery in the third set of trials [49] analysed the effect which staff experience has on productivity. Remembering that productivity declines with increasing staff levels, the results indicate that the productivity cost is higher for less experienced teams. In some cases organisations were adding less experienced staff to a project, resulting in it taking longer and costing more.

This study highlights some of the complex relationships between the development attributes or cost drivers. For example, it is not possible to state that as the system size increases, the productivity is expected to increase by a specific amount, as the staffing level also has to be considered, and how it interacts with the first two attributes and possibly changes their interrelationship. If then another attribute such as for example staff experience is introduced the interrelationship complexity increases further as was discussed above, and this may result in a multi-dimensional non-linear model.

The software development environment, partly because of these intricate interrelationships is complex. Models developed by traditional regression techniques have sometimes not provided a satisfactory solution, and this has prompted the interest in artificial neural networks, with their complex pattern recognition and modelling capability.

2.5.4 Banker and Kemerer Study

Banker and Kemerer [9] reconcile the two opposing views regarding the presence of economies or diseconomies of scale in new software development. Returns to scale reflect the influence of size as a cost driver. Increasing returns to scale result if at a given volume or size level, the marginal returns of an additional unit of input exceed the average returns. Economies of scale are thus present when average productivity is increasing, and diseconomies when average productivity is decreasing.

A reason for possible economies of scale is the use of sophisticated development tools [12]. These tools may increase productivity, but are typically expensive, both in the initial capital investment, as well as in the organisational learning curve. Small projects may make such a large cost not economically viable, suggesting that these sophisticated tools will more likely be used on large projects, who then benefit from the increased productivity. The larger projects may also benefit from staff specialisation [9], where a high level of expertise in specific areas may increase the overall project productivity.

Diseconomies of scale on the other hand may result from many other development environment attributes. Research has shown some correlation between system size and development team size [119], and it appears that larger projects are often developed with larger project teams. As the team size increases the number of communication paths increases at an increasing rate. Conte, Dunsmore and Shen [16] suggest that the number of communication paths required will be equal to $\frac{n(n-1)}{2}$. Such a non-linear communication cost overhead increase could contribute to diseconomies of scale [15].

Apart from increased communication paths, increasing team members may increase the possibility of conflict between team members, and this may reduce productivity [12]. Jones [57] on the other hand notes that many overhead activities, such as planning and documentation, also grow at a faster than linear rate as project size increases. The increasing effort may contribute to reduced productivity. Banker and Kemerer [9] suggest that project slack, which is more likely to appear on a larger project, may impact negatively on productivity. Conte, Dunsmore and Shen [16] point out that larger systems face increasingly complex interface problems between system components, which would increase development effort, thereby reducing productivity.

Generally development effort estimation models have the following form [16]:

$$y = a(x)^b$$

where y is the estimated development effort, and x a measure of the size of a project. The exponent b indicates a model's suggested response to scale. If b is less than 1 this indicates the presence of economies of scale. In contrast an exponent b greater than 1 indicates diseconomies of scale. In the Banker and Kemerer study [9] nine data sets are used to develop models of the above form. The exponent coefficient, b, for the various models is shown in Table 2.13.

Table 2.13 Summary of Log-Linear Models [9]

Survey Set	No. Projects	b
Behrens	22	0.94
Walston	60	0.91
Bailey	19	0.95
Yourdon	17	0.72
COCOMO	63	1.11
Albrecht	24	1.49
Belady	33	1.06
Wingfield	15	1.06
Kemerer	17	0.85

The exponent component, b, is not consistent and indicates economies of scale for the Behrens, Walston, Bailey, Yourdon, and Kemerer survey sets, and diseconomies of scale for the COCOMO, Albrecht, Belady, and Wingfield survey sets. Banker and Kemerer [9] reconcile this apparent inconclusiveness by suggesting that projects may exhibit both increasing and decreasing returns to scale, depending on the size of projects. They note that the simple log-linear model used does not allow for the possibility of increasing return for some projects and decreasing returns for others.

To attempt to resolve this problem and test the hypothesis that average productivity is increasing for projects smaller than the most productive scale size and decreasing for projects larger than the most productive scale size, the log-quadratic as well as quadratic models were developed for all the survey sets. The results were not

consistent in that these flexible parametric functional forms frequently violate reasonable regularity conditions. For example the log-quadratic models estimated for the Bailey and Wingfield data sets exhibit decreasing labour requirement for increasing project size for smaller systems. Similar violation of this regularity condition is exhibited by the quadratic models, for smaller systems in the Albrecht and Kemerer data sets, and for large systems in the Bailey, COCOMO, and Belady data sets.

Another, and probably more serious problem reported by the authors is the multicollinearity of the independent variables. This high level of collinearity implies that the confidence of interpreting the estimates of the coefficients of the independent variables will be very low for both the log-quadratic and the quadratic models, and as a result the estimates of these coefficients are likely to be unstable.

Banker and Kemerer [9] come to the conclusion that the usual econometric methods may not be appropriate for estimating the nature of returns to scale or the productive scale size for the data sets which they had included in their study. They comment that, 'Given these problems, and the limited *a priori* knowledge about the functional form of the production process underlying software development, specifying a parametric form for the production correspondence is difficult to substantiate theoretically or validate statistically. Also, it is not immediately apparent what restrictions these hypotheses, treated as axioms in the econometric approach, impose on the correspondence'. They propose the use of Data Envelopment Analysis (DEA), a non-parametric approach to production frontier estimation. DEA does not impose a parametric form on the production function and assumes only that a monotonically increasing and convex relationship exists between inputs and outputs.

Using DEA the most productive scale size was determined. Projects smaller than the most productive scale size present increasing returns, while projects larger than the most productive scale size present decreasing returns to scale. The use of the DEA technique thus facilitated the determination of the most productive scale size. The results suggest that the most productive scale size varies widely across different application environments, and an interesting extension to this research study would be the identification of the factors which contribute to some organisations' ability to successfully manage larger projects [9].

This study highlighted the problems associated with traditional statistical techniques and some limitations of this approach. Despite the solutions being non-trivial, the problem at issue is only a small aspect of the whole development environment. In the above study only the effect of system size was studied in relation to development productivity. There are many other cost drivers which affect productivity [16]. What complicates matters is that the relationship may be non-linear [7] [49] [75], and that the attributes may be interrelated [63] [75], requiring a multi-dimensional perspective.

Artificial neural networks are recognised for their ability to provide good results when dealing with problems where there are complex relationships between inputs and outputs, and where the input data is distorted by high noise levels [92]. These characteristics justify the choice of neural networks as research topic for this project in a search of possible solutions for the complex software development effort estimation problems as highlighted by the Banker and Kemerer study [9] above.

2.5.5 Vessey study

Vessey [114] studied the effect of certain factors which are commonly held to affect program development effort and programmer productivity. A field study was conducted in which data was collected from three business organisations. The total sample size was 353 programs, with 103, 191, and 59 projects from each company respectively.

An analysis of the results indicates that size is the principal determinant of both program development effort and productivity. The results indicate that there is a significant positive relationship between effort and system size. The relationship was significant in all three organisations.

The results also reveal a positive relationship between productivity and system size. In no organisation did structured programming have an influence on the effort required to develop programs or on productivity. From the analysis the other significant factor was programmer skill in one of the three organisations, indicating for that organisation that less skilled programmers took longer to develop programs than more skilled programmers.

This study, as also other similar studies, did not directly assess the resultant program quality. It makes the implicit assumption that all programs satisfy management's requirements to the same degree. Vessey concludes that with the random selection of programs and a reasonable sample size there is no reason to suspect bias.

2.5.6 Conclusion

The effect of various software development attributes on effort and productivity has not been found to be consistent and conclusive, and this complicates effort estimation model development. Some results of studies which have been reported in the literature were discussed in this section.

Further studies are required to identify and quantify the development attribute effects as they form an integral part of traditional development effort estimation models, and without them accurate estimation will not be consistently possible. Some problems which are encountered with traditional regression analysis were also discussed, as well as the justification of the use of neural networks for this thesis, for estimation in the complex software development environment.

2.6 Estimation Model Performance

2.6.1 Introduction

Unsubstantiated claims have been made regarding model prediction accuracy. Dreger [23] states that for Function Point Analysis (FPA) the Mean Absolute Relative Error (MARE) for estimates of existing systems is less than 10 percent. He also claims that FPA has a *proven* accuracy of a MARE of less than 20 percent for planned systems.

On the other hand some researchers have been critical of some existing models. Kitchenham [63] in her study has found that the exponential coefficients used in many models could not be shown to be significantly different from 1, and that the non-linear relationships which they assumed could not be substantiated. This questions the

appropriateness of the models and their coefficients. In her research into the technology factors of FPA her finding is that the 14 factors could be replaced by six new factors. In FPA the 14 technical adjustment factors are also treated as if they are independent and additive. Kitchenham concludes that the factors are not independent and that the technical adjustment may be inappropriate.

2.6.2 Development Effort Estimation Studies

In a study Finnie and Wittig [122] reviewed the performance of software effort estimation models as reported in some research publications. They evaluated how well the productivity function was modelled to derive accurate effort estimates from the size measures. Some of these findings are discussed in the following six sections.

2.6.2.1 Kemerer Study

Kemerer [60] presents an empirical validation of four algorithmic models. These are SLIM, COCOMO, Estimacs, and Function Points. SLIM and COCOMO are Source Line Of Code (SLOC) models, while the other two use function counts as their input. SLIM and Estimacs are proprietary models, while the other two are in the public domain.

Data was collected for each of the models from the same projects. This data was collected on projects outside the original model development environments, and the models were not re-calibrated to the new project data. The results therefore indicate to what extent these model are generalisable to environments other than those in which they were originally calibrated.

For all of the models data was collected on 15 projects, except for Estimacs for which data was collected on 9 projects. All projects were from a single organisation. To enable a comparison to be made to compare model prediction errors, only the 9 projects common to all models were used in the analysis, and the results presented here thus differ slightly from those in Kemerer [60].

Table 2.14 Model Error Comparison

	MRE	MARE	Std Dev	Min	Max
RE% SLIM	787.95	787.95	682.78	21.33	2221.29
RE% COCOMO	357.44	357.44	241.27	83.26	825.69
RE% Estimacs	67.89	85.48	89.37	6.67	198.74
RE% FPA	22.61	56.96	71.82	11.67	120.98

The results are shown in Table 2.14. In all cases the relative error percent (RE%) is shown. The results are not encouraging and show large estimation errors. As can be seen from Table 2.14, the Mean Absolute Relative Error (MARE) ranges from an average of 57 percent to almost 800 percent. The standard deviation is very high as is reflected in the large range of prediction errors between the minimum error and the maximum error. To note is that most of these models show a strong bias. This is reflected by the magnitude of the error and the small difference between the Mean Relative Error (MRE) and MARE. In almost all cases the models largely overestimated the development effort.

An analysis by this author was made of the prediction error if the bias is removed. The results are shown in Table 2.15.

Table 2.15 Model Comparison After Bias Removal

	Bias coefficient	MRE%	MARE%	Std Dev	Min	Max
RE% SLIM	0.1126	0	56.39	76.89	5.84	161.42
RE% COCOMO	0.2186	0	42.86	52.74	14.46	102.36
RE% Estimacs	0.5956	0	45.39	53.23	16.15	77.93
RE% FPA	0.8156	0	44.13	58.58	2.16	103.21

The bias coefficient shows the adjustment factor applied to each model. As can be seen, in all cases there was a positive bias which was removed by a factor smaller than one. SLIM required the largest adjustment, while Function Point Analysis needed the least with only approximately a 20 percent bias adjustment. Because the bias has been removed the MRE is 0, as the positive and negative errors cancel each other. With the bias removed the differences between the prediction accuracy of the models is greatly reduced. The results for COCOMO, Estimacs, and Function Point are very similar, with a MARE of about 45 percent. The error of the SLIM model is slightly higher. The standard deviation follows a similar trend.

These results indicate that the models did not generalise well outside their original development environment. This result appears to corroborate the finding of other researchers such as Jeffery [50] and Kusters et al [66]. Models which are not calibrated to the local development environment produce less accurate estimates than models which have been re-calibrated.

2.6.2.2 Ferens and Gurner Study

In this study [31] three development effort prediction models were evaluated. All three accept function points as their input. The data used for this study were 22 projects from Albrecht's database, and 14 from Kemerer's data set. Thirty of these thirty six projects were written in COBOL. The models' results are shown in Table 2.16.

Table 2.16 Statistical Test Results

Model	MARE	Std Dev	Within +/- 30%
SPANS	0.62	0.53	0.36
Checkpoint	0.46	0.53	0.47
COSTAR	1.05	0.84	0.19

The prediction error is large, with the MARE ranging from 46 percent for the Checkpoint model to 105 percent for the COSTAR model. Such large errors place in doubt their use as management planning and control tools. In all three models the standard deviation is high and in the COSTAR model less than 20 percent of the

estimates were within plus or minus 30 percent of the actual effort required to develop a project.

Table 2.17 Statistical Test Results After Bias Adjustment

Model	MARE	Within +/- 30%	Within +/- 20%
SPANS	0.34	0.50	0.33
Checkpoint	0.32	0.56	0.39
COSTAR	0.34	0.50	0.31

The authors determined by means of the Wilcoxon 'T' Statistic that the models were biased and that all of them tended to over-estimate effort. When the bias is removed the results improve, as can be observed in Table 2.17, although the errors are still large. Despite the bias being removed the models predicted within plus or minus 30 percent of the actual effort in only approximately 50 percent of the cases. In all cases model performance as measured by the MARE is similar.

2.6.2.3 Jeffery and Low Study

Jeffery and Low [50] conducted a study to investigate the need for model calibration at both the industry as well as the organisation level. In this study the CLAIR estimation package was used. The algorithms used in CLAIR are either based on Source Lines of Code (SLOC) or function point counts. The estimation package is also able to convert function point counts into SLOC. Data was collected from three companies, called here A, B, and C. For each of these companies data was gathered on seven projects.

An analysis was made comparing the prediction accuracy of CLAIR using SLOC as input, and function points as input, to try and determine which is the more appropriate input size measure. The results are shown in Table 2.18.

Table 2.18 FP and SLOC Model Comparison

Detail	A	B	C
SLOC-based MRE%	115	69	27
FP-based MRE%	39	105	-32
SLOC-based MARE%	117	77	43
FP-based MARE%	53	105	43

In Company A the MRE and the MARE are lower in the FP based model, while for Company B the opposite is true. In Company C the prediction error is of similar size for both metric types. The results are thus inconsistent and inconclusive.

In the study by Jeffery and Low an analysis is made of function point to SLOC conversion ratios in various companies. The conclusion is that SLOC per function point ratios are not consistent across organisations, even for the same language (see Section 2.4.2.1). This implies that the backfire process of converting function points to SLOC for estimation purposes as is carried out in some generic models [56] cannot be generally supported [50].

Using SLOC as input, an analysis is made of the effort estimation error of CLAIR. The results are shown in Table 2.19. The results are shown for each company individually, and also the average result for all three companies combined. The high MARE is noted. Company C, which had the lowest estimation error reflects a MARE of 43 percent. The combined MARE is almost 80 percent, and it is difficult to see how models with such large MARE can be used with confidence in development performance determination. The large MRE and MARE suggests a bias in the model, which in this study generally tended to over-estimate the development effort. The effectiveness of the model when the bias is removed is investigated by this author and the results are shown in Tables 2.20 and 2.21.

Table 2.19 CLAIR Effort Estimation Error

Company	MRE	MARE	Std Dev	Min	Max
A	115.28	117.51	72.22	7.78	194.43
B	69.80	77.54	88.22	8.12	214.87
C	26.95	43.24	46.77	14.97	91.80
Combined	70.68	79.43	76.95	7.78	214.87

Table 2.20 Results with overall bias adjustment

Company	MRE	MARE	Std Dev	Min	Max
A	26.13	44.14	42.31	17.07	72.51
B	-0.51	40.93	51.69	10.03	84.48
C	-25.62	29.95	27.40	2.77	58.12
Combined	0.00	38.34	45.09	2.77	84.48

In Table 2.20 the model bias was removed across all three companies combined. The adjustment coefficient was 0.59. There is a reduction in MARE, from 79 percent to 38 percent. Even a model with a 38 percent MARE will not allow for great confidence in the productivity measure, especially when the standard deviation is 45 percent, and the maximum estimation error is as high as 84 percent.

In Table 2.21 the bias adjustment is made for each company individually. The adjustment coefficients for each company are shown. There is a further slight reduction in the overall estimation error, but the MARE is still high at 32 percent, with the standard deviation just less than 40 percent and the maximum error of 85 percent.

Table 2.21 Results with individual organisation adjustment

	Bias	MRE	MARE	Std Dev	Min	Max
A	0.46	0.0	26.1	33.6	5.2	57.2
B	0.59	0.0	40.9	52.0	9.6	85.4
C	0.79	0.0	29.0	36.8	4.7	51.1
Combined		0.0	32.0	39.4	4.7	85.4

2.6.2.4 Mukhopadhyay et al Study

Mukhopadhyay et al [78] state that existing algorithmic models fail to produce accurate software effort estimates. A new model (Estor), based on case-based reasoning is evaluated against expert judgement, COCOMO, and Function Point Analysis (FPA). The case-based reasoning approach [65] [97] [100] [118] stores previous estimates with the relevant attributes as cases in its case memory database. For each new estimate it first searches its case-base to establish whether any cases similar to the one under consideration have previously been stored. Using similarity metrics the 'most similar' previous case is extracted and used to solve the current problem. The relevant features of this case are then also stored in its case-base to assist in improving future estimation performance.

For this study the Kemerer data set [60] was used. The expert judgement and the Estor model are compared to the FPA and COCOMO results from Kemerer's study. The results are shown in Table 2.22.

The estimation error of the expert and Estor are considerably less than FPA and COCOMO. Despite the improvement the errors are still large. Unfortunately only one expert was used in this evaluation and it is not clear how representative his level of judgement is of the expert judgement approach in general. Estor's MARE was greater than 50 percent, with a standard deviation of 38 percent, and a maximum error of 107 percent. Of concern is that none of the models were able to improve on the performance of the expert.

Table 2.22 Model Error Results

Model	MARE %	Std Dev %	Max	Min
Expert	30.72	21.74	72.41	0.86
Estor	52.79	37.92	106.90	0.98
Function Point	102.74	116.05	326.72	0.23
COCOMO	618.99	680.83	2685.34	106.97

2.6.2.5 Heemstra Study

Heemstra [40] states that software cost estimation models are not generally accepted in the organisations which he surveyed, and that only 51 of the 364 organisations which estimate software development effort use models. His analysis shows that these 51 model users make no better estimates than the non-model users. Heemstra's finding is that most of the time generic models are used without re-calibration, and that most models do not support re-calibration. He states that if models cannot be adapted to the local development environment they will produce less accurate estimates.

The data for Heemstra's study was the result of 14 experienced project leaders making an estimate for a project that had actually been completed. Three approaches were evaluated. These are the expert estimate, the BYL (Before You Leap) model estimate, and the Estimacs model estimate. The BYL and Estimacs models were the only two models which met certain model selection criteria which were set by Heemstra.

BYL is a commercial package based on a link-up between FPA and COCOMO. It starts with function points and translates these into SLOC, using various conversion factors for different programming languages. Estimacs is a proprietary model which uses a function count as its input. It is not clear how it translates this input into an estimation of effort. As a final comparison, the experts were shown the estimates of the models and the other experts, and were then requested to make a final estimate.

The estimates for the project are shown in Table 2.23. The actual project took 8 person-months (PM) to develop. The difference between the estimated and the actual effort is alarmingly large, leading Heemstra to conclude that on the basis of such

discrepancy the selected models have not shown that they can be used as a reliable estimation tool at an early stage of a project development.

Table 2.23 Results of Heemstra Study

Estimation Method	PM (mean)	Std Dev
Experts	28.40	18.30
BYL	27.70	14.00
Estimacs	48.50	13.90
Final Estimate (Experts)	27.70	12.80

2.6.2.6 Jeffery, Low and Barnes Study

Jeffery et al [53] state that one of the fundamental problems associated with development effort estimation is the *a priori* estimation of software size. Although FPA has increased in popularity over the last decade, a number of criticisms have emerged over this period. Most of them relate to the manner in which function counts are calculated, and the impact of processing complexity. SPQR/20 is a model developed by Jones [56] who claims to have overcome these deficiencies. Jeffery et al [53] in their study compare the SPQR/20 model to FPA.

The data was all collected within one organisation, and was collected from 64 projects. Three of them were written in COBOL and 61 in PL/1. The models were calibrated to the local environment using regression analysis to determine the coefficients. The results are given in Table 2.24.

Table 2.24 FPA/SPQR Comparison

Method	MARE %	Std Dev
FPA	11.93	10.1
SPQR/20	12.26	10.3

The estimation errors are considerably less than those of previous studies and reflect the benefits of model calibration to the local environment. The results indicate that there is no significant difference between the estimate of the two approaches and in both the FPA and the SPQR/20 method the processing complexity adjustment does not affect the accuracy of the effort estimates. This conclusion validates the comments of Kemerer [60] and Symons [103] regarding the usefulness of the 14 processing complexity factors. Jeffery et al conclude that the 14 processing complexity factors in FPA do not provide a useful input to the FPA method, and neither does the SPQR/20 complexity factor.

2.6.3 Conclusion

Large resources are spent on software development effort each year. It has been estimated that in 1990 the cost of software development and maintenance was $US250 billion [13]. Relatively small increases in productivity could therefore result in large savings in information systems expenditure. A necessary prerequisite to making confident decisions based on productivity measures is the accurate determination and modelling of the productivity function.

The models reviewed indicate that current generic models have not been able to estimate with sufficient accuracy to detect small improvements in productivity. The models which were studied were not able to generalise well outside their development environments, and this finding concurs with Jeffery [53] and Heemstra [40] who state that uncalibrated generic estimation tools are unlikely to make accurate predictions. Heemstra suggests that current models are no better than expert judgement.

2.7 Contribution of this Study

The software development environment was reviewed in general in this chapter, with a particular focus on those issues which affect software development effort estimation. Some problems which are inherent in the size metric itself were discussed, as were problems in its application. In Chapter 1 the importance of accurate size estimation for

project planning and control was highlighted, and in this chapter the performance of some published effort estimation results were reviewed.

Kusters et al [66] in their study noted that current model estimates have not shown to be better than manual estimates. They also note that despite the flood of publications on software estimation models, they are not aware of any empirical study that shows the ability of software cost-estimation models to predict effort of projects accurately. This disappointing conclusion reflects the complexity of the problem of effort estimation, as on initial investigation it deceptively appears not to be a particularly difficult task. Such disappointing results though do not reflect a lack of competence and intellectual ability of researchers working in this area.

There is a large incentive to develop accurate estimation models, but despite considerable effort and intellect being focused on this issue, further understanding and progress needs to be made to enable software metrics and estimation models to be used generally and confidently in the planning and control of software projects.

The contribution of this chapter was to investigate several aspects of the problem of software development effort estimation. Several issues were identified which contribute to the complexity of estimation and which may be a source of problems for the neural network models. The above sections also reviewed benchmark performance standards against which the neural network estimation model performance may be assessed.

Chapter 3

Artificial Neural Networks

3.1 Introduction

Artificial Neural Networks (ANNs) are recognised for their ability to provide good results when dealing with problems where there are complex relationships between inputs and outputs, and where the input data is distorted by high noise levels [112]. The software development environment from which development effort estimates are generated, are characterised by these attributes.

Banker et al [7] suggest that linear models are likely to be inadequate representations of the development process. ANNs have the ability to model non-linear relationships and are capable of approximating any measurable function [46]. Networks are thus universal approximators, and this implies that any lack of success in applications must arise from inadequate learning, insufficient numbers of hidden units, or a lack of a deterministic relationship between input and target [46].

ANNs have several features which make them attractive prospects for solving pattern recognition tasks without having to build an explicit model of the system. In a broad sense the network itself is a model because the topology and transfer functions of the nodes are usually formulated to match the current problem. Many network architectures have been developed for various applications and for estimating software development effort in this project, back-propagation networks and cascade networks

are used. A basic discussion of neural networks and the structure of neurons, as well as the topology and operation of back-propagation networks is given in Appendix B.

Classical back-propagation [92] is a gradient method of optimisation executed iteratively, with implicit bounds on the distance moved in the search direction in the weight space. This is achieved by incorporating a learning rate (the gain) and the momentum term (the damping factor) in the model.

The performance of neural networks depends on the architecture of the network and their parameters. Determining the architecture of a network (size, structure, connectivity) affects the performance criteria, such as the learning speed, accuracy of learning, noise resistance and generalisation ability [74]. There is no clearly defined theory which allows for the calculation of the ideal parameter settings and as a rule even slight parameter changes can cause major variations in the behaviour of almost all networks [99].

This research project studied the use of back-propagation ANNs and cascade networks for software development effort estimation. For one dataset the k-nearest-neighbour (k-NN) approach is also used to predict development effort and the estimation error is compared to that of the neural networks. The network inputs were various development attributes and system size measured in function points. The output was predicted development effort (development hours). These results were studied for consistency and accuracy by determining the Mean Relative Error (MRE), and the Mean Absolute Relative Error (MARE).

3.2 Neural Network Learning

Training or learning in a neural network requires that the network be run for numerous iterations or epochs, each time adjusting the connection weights.

3.2.1 Varieties of Learning Procedures

Numerous learning procedures have been identified [44]. The most popular ones are supervised learning, reinforcement learning, and unsupervised learning.

With supervised learning the network has an input and an output layer, as well as zero or more hidden layers. The actual output is compared to the desired output and the difference or error is used to adjust the weighted connections in an effort to improve the network performance by reducing the error on the next epoch or training iteration. The neural networks which were used for this thesis use a supervised learning procedure, and this is discussed below in further detail.

With reinforcement learning the network topology is similar to supervised learning but there is now a global reward system, but this approach is very slow as it needs many trials to assign credit to the weights correctly.

With unsupervised learning the network only has the input units and hidden units. The hidden units model the higher-order statistical structure of the set of input vectors [44].

3.2.2 Back-Propagation

Back-propagation is a supervised learning procedure. The difference between the actual and desired output is determined continuously and repetitively for each set of inputs and corresponding set of outputs produced in response to the inputs. A portion of this error is propagated backward through the network. This is shown graphically in Figure 3.1. At each neuron the error is used to adjust the weights of the connections so that for the next epoch the error in the network response will be less for the same inputs. This process continues until the network performance on the test set is optimised.

The implementation of the back-propagation algorithm varies slightly for various networking tools. For the Neuralyst [80] tool, which is a typical back-propagation tool, learning and the adjustment of weights is accomplished as follows.

The error based weight adjustment is influenced by two factors, the learning rate (LR) and the momentum factor (M). The learning rate is set in the range between zero and one. This determines the proportion of the calculated error based weight adjustment which is actually applied to the connection weight. If the learning rate factor is set to a large value, the network may learn more quickly, but depending on the application there is a real danger that this may cause it to over-correct constantly causing a large

variability in the input set. With these oscillations across the error plane the network may not learn well at all [43] [108].

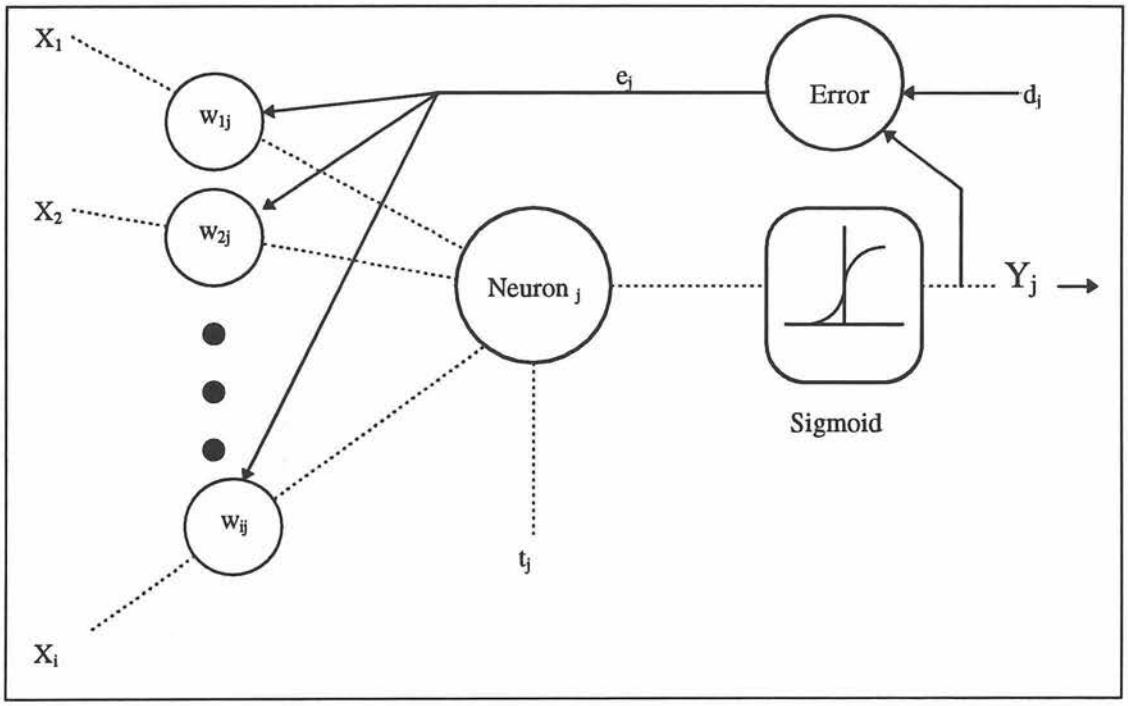


Figure 3.1 Error Back-Propagation

If the error rate is set at zero the network will not learn at all, as none of the calculated adjustment is applied, whereas a learning rate of one will apply the full adjustment. In practice it is usually better to set the factor to a small value and progress slowly toward the target. If the error rate is set to a low value it may cause the network to get stuck in local minima. Empirical evidence suggests that these local minima are usually close to the global minimum, and unless absolute network optimisation is required for an application, this may not be particularly onerous [44].

The momentum or weight decay factor on the other hand allows a change in weights to persist for a number of adjustment cycles. This determines the proportion of the previous cycles weight adjustment which is applied to the current weight adjustment. Again the momentum may be set between zero and one. Some persistence of previous adjustments may improve the learning rate in some situations, by helping to smooth out unusual conditions in the training set [80] [107].

The learning process begins at the output layer as is shown in Figure 3.1, utilising the following equation:

$$w_{ij} = w'_{ij} + LR \times e_j \times X_i$$

where $e_j = Y_j \times (1-Y_j) \times (d_j - Y_j)$

For this process the *i*th input of the *j*th neuron in the output layer, the weight w_{ij} is adjusted by adding to the previous weight value, w'_{ij}, a term determined by the product of the learning rate, LR, an error term, e_j, and the value of the *i*th input, X_i. The error term, e_j, for the *j*th neuron is determined by the product of the actual output, Y_j, its complement, $1 - Y_j$, and the difference between the desired output, d_j, and the actual output.

Once the error terms have been computed and the weights are adjusted for the output layer, the values are recorded and the weights to the preceding layer are then adjusted. These weight adjustments are similar to the process for the output layer, except that the error term is now calculated slightly differently, as the succeeding layer adjustments have already been made. For this adjustment the following equation is used:

$$e_j = Y_j \times (1-Y_j) \times \Sigma(e_k \times w'_{jk})$$

The difference between the desired output and the actual output is now replaced by the sum of error terms for each neuron, *k*, in the succeeding layer, times the respective pre-adjustment weights [80].

With the introduction of the momentum factor the $w_{ij} = w'_{ij} + LR \times e_j \times X_i$ equation from above is slightly adapted to:

$$w_{ij} = w'_{ij} + (1-M) \times LR \times e_j \times X_i + M \times (w'_{ij} - w''_{ij})$$

Now the momentum factor (M), the previous weight, w'_{ij}, and the next to previous weight, w''_{ij}, are included in the calculation.

As the network trains and it starts converging, the error rate reduces. The decision of when to stop training, as well as the dangers of over-training are discussed later.

3.3 Network Issues

Various issues have to be resolved before neural networks can be implemented to solve a problem. Hertz, Krogh and Palmer [43] summarise some of these as follows:

- 'What is the best architecture? Should the units be divided into layers or not? How many connections should be made between units, and how should they be organised? What sort of activation or transfer function should be used? What type of updating should be used: synchronous or asynchronous, deterministic or stochastic? How many units are needed for a given task?'
- 'How can a network accomplish the task of learning? Should it learn a task or must it be pre-designed? If it can learn a task, how many examples are needed for good performance? How many times must it go through the examples? Does it need the right answers during training, or can it learn from correct/incorrect reinforcement? Can it learn in real-time while functioning, or must the training phase be separated from the performance phase?'
- 'What can the various types of network do? How many tasks can they learn? How well? How fast? How robust are they to missing information, incorrect data, and unit removal or malfunction? Can they generalise from known tasks or examples to unknown ones? What classes of input-to-output functions do they represent?'

These are some of the issues which were addressed in selecting the network architecture and topology. As this research project is not primarily concerned with the theory of artificial neural networks, these issues are not discussed here in detail. Hertz, Krogh and Palmer [43], and Rumelhart, McClelland and the PDP Research Group [94] have further details on these issues.

3.4 Network Parameter Settings

In this section some of the problems of finding the best network topology and setting the corresponding parameters are discussed. Mandischer [74] reports that the performance of neural networks depends on their architecture and the parameter values that were used. As there are only very few mathematical methods to determine the

architecture and parameters of networks, design decisions are normally achieved by rules of thumb. The common approach to designing neural networks is thus to build a network, and test it for the desired function. The network is then refined by changing the structure and parameters and assessing the network until the evaluation criteria have been met [74].

Schoneburg [99] noted that as a rule even slight parameter changes may cause major variations in the behaviour of networks and that there is no theory yet available which can be used as a general guideline to consistently find the best network.

The momentum coefficient must be between zero and one [80]. A value of 0.9 is often chosen [43], but this may not give the best results for all applications. Tan and Wittig [108] studied the interaction of learning rate and momentum, and for the data that was used, which was the Ford Motor Company stock on the New York Stock Exchange, a learning rate of about 0.3, with a momentum of 0.7 gave the best overall results. Freisleben [34] also in a stock price prediction application found that a momentum of 0.7 gave consistently good results with a learning rate of 0.7 to 0.9.

The following guidelines to using back-propagation networks, which were developed from extensive experience are given by Hinton [44].

- Get as big a training set as possible as 'several' bits per weight are required.
- Initially try a net without hidden units, as the relationship may be linear.
- Use cross-validation to decide when to stop learning.
- Bias units (thresholds) should preferably have an adjustable bias. Typically the bias is just a normal weight whose input is constant and usually equal to one.
- Use symmetric activation functions such as sigmoid with a range of say -1 to +1, as convergence is usually faster than an activation such as the logistic function with a weight range of 0 to 1.
- Use target values within the range of the sigmoid function. If target values for the desired outputs are equal to the asymptotes of the activation functions, as for example -1 to +1, it has undesired effects. Target values on the symptotes tend to

drive the weights to infinity, slow down learning, and worsen generalisation. Target values of -0.7 to +0.7 with the sigmoid function are preferred.

- The initialisation weights should be adapted to the application. Random weights should be smaller for units with many inputs, and larger for units with few inputs. This keeps the expected weighted sum within the range of the sigmoid. On the other hand initialisation weights should not be too small, as this can cause the gradients to be very small and the learning initially to be very slow. On the other extreme the weights should not be too big, as that might saturate the units and produce tiny gradients. Large learning rates have a similar effect.
- The input values should be normalised in the range of 0 to 1 for best results. Large order of magnitude differences between inputs, and between inputs and the target values deteriorate the network performance and may make it difficult for it to converge [21].

The above are generalised guidelines. Different applications have different characteristics. Systematic trials with data will indicate the network behaviour for a specific set of data. This will identify more promising areas within certain parameter coefficient ranges, which will then have to be explored more intensively.

3.5 Generalisation and Over-Fitting

The network parameter values directly affect the generalisation ability of networks. These parameters include the network topology and connectivity, the number of hidden units for each layer, the range of initial random weights, the parameters discussed in the previous section, as well as over-fitting [94]. The operational definition of over-training or over-fitting is that the error on the test sample increases if training continues for too long [70]. It is important that training be stopped at the appropriate point, and not be allowed to proceed after the training error is as small as possible.

An explanation given by Ling [70] of over-fitting is that if data contains a primary regularity as exhibited by the majority of training examples, and a secondary regularity, then a network of appropriate size and training would at a certain point capture most

of the primary regularity, before picking up the secondary regularity. Such a network will have the best generalisation ability. This would be reflected by the lowest prediction error in the test data. With further, and thus excessive training, the network will start to curve-fit, and fit the secondary regularity as well. This will result in a deteriorating generalisation ability and an increasing prediction error in the test set will show this.

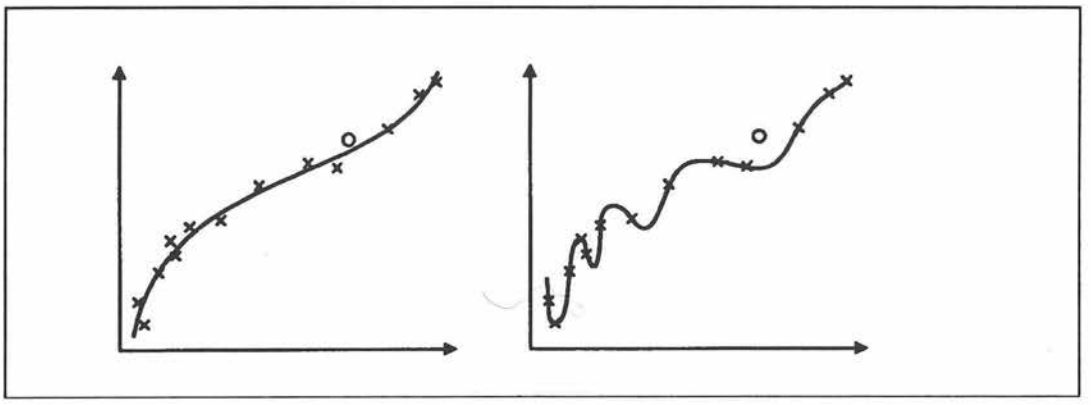


Figure 3.2 Primary and Secondary Regularity Curve Fitting

The problem of over-fitting is shown diagrammatically in Figure 3.2. The X points are the training input set, while the single O is the single point in the test set. The graph on the left shows a good fit to noisy data. With further training curve-fitting will result, and this is shown in the graph on the right in Figure 3.2. The error in the training set will be lower, but is likely to be poor on the test set, as is represented by the circle in the graph.

To prevent over-fitting the neural network is trained and at frequent intervals the test prediction error is calculated and monitored. Typically as the network starts training the error starts reducing, as will the test set prediction error. When this error does not improve any further, ie when it is at its minimum, network training should cease, as at this point the network will exhibit its best ability to generalise. The training error is not important here and this may often show considerable improvement with further training.

Another important lesson about generalisation is learnt from statistics and curve-fitting, where too many free parameters result on over-fitting. A curve fitted with too many parameters will follow the small detail or noise, but is poor for interpolation and extrapolation. The same is true for neural networks, where too many weights give poor generalisation [43].

This highlights one of the major advantages of cascade networks, as they automatically build a near minimal network and this assists in providing the good generalisation capability which they exhibit [28]. With back-propagation networks the guideline is to start training with only a few hidden units and measure the test prediction error continuously. The number of hidden units are then gradually increased, while each time the minimum prediction error is monitored. While the prediction error improves the number of hidden units are increased. Adding any more neurons after this may improve the training set error, but the generalisation ability will start to deteriorate. [44].

A further problem with overly large networks is that they may not converge, and the training time is much longer. In back-propagation networks the appropriate size has to be determined by trial and error as there is no available algorithm to calculate this in advance [70]. In general, if two models fit the data equally well, the simpler one probably generalises better [44].

Another problem encountered when developing neural networks is determining whether the test set is large enough to permit the neural network to sufficiently capture the problem domain to facilitate good generalisation. Baum and Haussler [10] show that a net will almost certainly generalise satisfactorily if:

- the fraction of errors on the training set is less than $\frac{\varepsilon}{2}$
- and $m > \theta\left(\frac{W}{\varepsilon}\log\frac{N}{\varepsilon}\right)$

where: m is the number of training cases

N is the number of nodes (just one layer)

W is the number of weights

ε is the allowed fraction of errors on the test set (assume $\varepsilon < \frac{1}{8}$) [44]

According to Hinton [44], in practice all that is needed is for $m > \frac{W}{\varepsilon}$

It is thus important to try and have models with lots of training data. Just as in regression analysis, a large premium is placed on small models as training data is often limited.

3.6 Cascade Networks

3.6.1 Introduction

The Cascade-Correlation learning algorithm was first proposed by Fahlman [28]. It was developed in an attempt to overcome certain problems and limitations of the popular back-propagation learning algorithm [92]. Cascade-Correlation is a supervised learning architecture that dynamically builds a near-minimal multi-layer network topology during the training process [26]. Cascade-Correlation eliminates the necessity for the network user to guess in advance the network's size, depth, and topology [26]. This eliminates many of the problems discussed above.

The near-minimal network is built automatically, and even though the architecture initially appears complex involving many steps, it trains much faster than a back-propagation network [27]. In Cascade-Correlation the selected candidate hidden unit's input weights are frozen, the feature-detector, once built, is never altered or cannibalised, and so the network can be trained incrementally. A large data set is thus broken up into smaller 'lessons', and feature-building is cumulative [26].

The advantages of Cascade-Correlation are fast learning, good generalisation, and this is particularly important in software development estimation, automatic choice of network topology, ability to create complex high-order feature detectors, and the ability to learn complex behaviours through a sequence of simple steps [26]. For these reasons the decision was made to use the improved version of the original Cascade-Correlation network tool to complement the back-propagation network approach for this research project.

3.6.2 Cascade-Correlation Architecture

Initially the network contains only an input layer and an output layer, with the connections between them. As in back-propagation these units are fully connected. This single layer of connections is trained using the Quickprop algorithm [27] to minimise the error. Typically as with back-propagation, as the network starts training the reduction in error is relatively large with each epoch. As training progresses this improvement reduces, and approaches an asymptote. When no further improvement in the level of error is detected, the network's performance is evaluated. If the error is small enough and within the pre-set limit, training stops. If this error has not reached this level a hidden unit is added to the network in an attempt to reduce the residual error [45].

The selection and inclusion of a hidden unit into the network is done in the following manner. A pool of candidates is created. The number of candidate units generated depends on the application, but 10 should suffice in most cases [21]. Each candidate receives the weighted connection from the network's inputs, and from any hidden units already present in the net. At this stage the outputs of these candidate units are not connected into the active network.

This pool of candidates is then trained. As in the previous case this sub-net of candidates is only a two layer network. With multiple passes through the set each candidate unit adjusts its incoming weights to maximise the correlation between the output and the residual error in the active net. When the correlation stops improving the best candidate from the pool is chosen. Its incoming weights are then frozen and this unit is then added to the network. This process is called 'tenure' [45]. After tenure the unit becomes a permanent feature detector in the net, and all the unsuccessful candidates are discarded.

Since the new hidden unit receives connections from the previously added hidden units, each hidden unit effectively adds a new layer to the network. This is shown in Figure 3.3 [45]. This shows the architecture after two hidden units have been added. The vertical lines sum all the incoming activation, while the boxed O connections represent the frozen weights, and the boxed X connections are trained repeatedly.

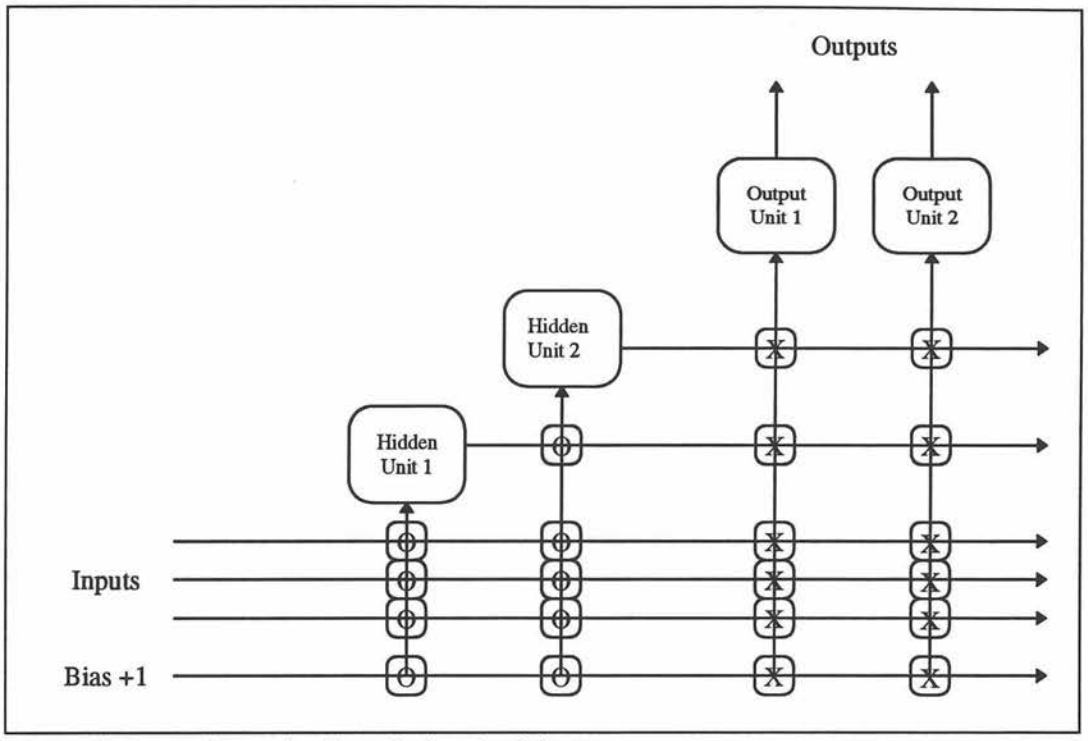


Figure 3.3 Cascade-Correlation Architecture

The next step is to re-train all the weights going to the output units, including those from the newly added hidden unit. With each epoch again the residual error reduces, typically at a declining rate. When there is no further improvement training stops and the network is again evaluated. If the residual error is small enough, or there is no improvement in this error since the addition of the last hidden unit, network training is stopped altogether. If this is not the case, the process of adding another hidden unit from a candidate pool is repeated.

Cascade-Correlation learns faster than back-propagation. One reason is that at any given time only a single layer of weights is being trained. There is thus never any need to propagate error information backwards through the connections, and the dramatic slowdown that is typical when training back-propagation nets is avoided [45]. Another factor is that the Cascade-Correlation algorithm is a 'greedy' algorithm, as each unit grabs as much of the residual error as it can. In a standard back-propagation network all the hidden units are changing simultaneously, thus all competing for the work that has to be done [45]. This is slow and unreliable.

An improved version of the original Cascade-Correlation network software has been developed by Fahlman. This version is called Cascade2 [25]. The improvements are in the area of training the candidate units. The covariance used in the original Cascade-Correlation networks tends to overshoot small errors, which is only a problem for analogue outputs.

Cascade2 instead tries to train candidate units to minimise the sum-squared difference between the scaled unit outputs and the residual error. Since there is a different trainable scale factor for each output, this ends up looking just like a layer of a back-propagation network, with candidate input and candidate output weights and a single hidden layer. Once the candidate units have been trained, their selection and installation are as in the original Cascade-Correlation network [25].

According to Fahlman [25], Cascade2 appears to work much better for continuous output tasks, and also some binary output tasks, where correlation training tended to get stuck and did not want to converge. Even though Cascade2 has not yet (November 1994) been released for general distribution, as it had not been sufficiently tested, this research project decided to use this software, and correct any problems in the code as they are identified [21].

3.6.3 Cascade Parameters

To run the Cascade2 network, a network file is created in which the various parameter values are set. These settings may in some cases affect the network performance. This section briefly covers some of the main settings and is included to provide background information. The inclusion of this detail here makes some of the later discussion in Chapters 4 and 5 on the research design and development and analysis of the cascade models easier to read as the focus of that discussion remains on model building and analysis of results.

In total approximately 70 parameter values may be set. Most of these have default values which may be changed if the application requires this. For this research project most of the default values were accepted, except for up to about 20 parameters. These

were identified in consultation with Diederich [21], Szabo [105], and Fahlman [25]. Some of the more important ones are discussed below.

The network file in addition to setting the number of inputs, outputs, training patterns and test patterns also includes the following parameter settings (the spelling/writing convention is taken from the Cascade2 C source code):

- UnitType – This sets the transfer or activation function of the hidden units and can be set to either sigmoid or gaussian. Experience suggests that the sigmoid function performs better in most cases [21].
- OutputType – This sets the transfer or activation function of the output units and can be set to either sigmoid or linear. For problems in which a precise analog output is desired instead of a binary classification, linear output units may be the best choice [28].
- WeightRange – Here the range of the random initialisation weights is set. A value of 0.5 means that these values will be generated in the range -0.5 to +0.5. The cascade networks have the advantage that when faced with insufficient weight range they will not fail abruptly, but will instead add extra units to compensate for this deficit.
- UseCache – If this is set to TRUE the forward-pass values are cached instead of repeatedly computing them.
- Ncandidates – This determines the size of the candidate pool. In applications where it is very important that only the *best* candidate is selected, where for example precision is critical, a large pool of say 100 may be generated. This has the disadvantage that the network training time is slowed down considerably.
- ErrorIndexThreshold – Training stops when the ErrorIndex is less than the ErrorIndexThreshold. This value is dependent on the level of precision or optimisation which is required.
- OutputEpsilon – This controls the amount of linear gradient descent to use in updating output weights and is similar to the parameter setting learning rate used in the back-propagation models. A similar, and self explanatory parameter setting is CandOutputEpsilon.

- OutputDecay – This factor times the current weight is added to the slope at the start of each epoch and keeps the weights from growing too big. A similar and self explanatory parameter setting is CandOutputDecay.
- OutputPatience – This sets a limit on the number of epochs with no real change after which network tuning is stopped.

Some of these coefficient value settings were influenced by suggestions made by Diederich [21], Szabo [105], and Fahlman [25], while others were determined by the experience gained from working with the application.

3.6.4 Conclusion

Cascade networks eliminate some of the problems associated with back-propagation networks. Its main advantage is its automatic construction of a near-minimal multi-layered network topology. It is also possible to build networks with a mixture of non-linear unit-types as was described above.

Cascade networks learn fast. In back-propagation the hidden units engage in a complex dance before they settle into distinct useful roles. In Cascade networks each unit sees a fixed problem and can move decisively to that problem, and the learning time in epochs grows very roughly as $N \log N$, where N is the number of hidden units ultimately needed to solve the problem [28].

Cascade networks can build deep networks (high-order feature detectors) without the dramatic slowdown evident in back-propagation networks with more than one or two hidden layers [28].

As was discussed in section 3.6.2 the cascade network learns incrementally. This has the advantage that it is possible to add new information to an already trained network, as once built, a feature detector is never cannibalised. It is available from the time it is introduced for producing outputs or more complex features. Training on a new set of examples may alter a network's output weights, but these are quickly restored if the network is returned to the original problem.

At any given time only one layer of weights in the network is trained. The rest of the network is not changing and so the results can be cached. There is also never any need to propagate any error signals backwards through the network connections. A single residual error can be broadcast to all candidates. The weighted connections transmit signals therefore in only one direction eliminating the troublesome difference between back-propagation connections and biological synapses.

The candidate units do not interact with one another, except to pick a winner. Each candidate sees the same inputs and error signals at the same time. This limited communication makes the architecture attractive for parallel implementation [28].

3.7 Discussion

Multiple regression analysis and other statistical techniques have commonly been used to develop software development effort estimation models [16]. A pilot study (Chapter 4) indicated that artificial neural networks may provide alternative models to these conventional approaches. Artificial neural networks have been applied to a variety of problem areas. In many instances they have provided superior results to conventional methods. For example, Lang et al [67] have shown that an artificial neural network outperformed a continuous-parameter hidden Markov model on noisy word recognition tasks. Then Lippman and Gold [71] demonstrated that a three-layered network performed better than gaussian classifiers in speech recognition. In the business environment, statistical analysis methods, such as regression analysis and discriminant analysis have been widely used in forecasting, although the performance of these approaches has often been less than conclusive [124].

Recently artificial neural network techniques have been applied to business problems such as the prediction of bond ratings and stock price performance [24] [47] [102] [112]. These results have demonstrated that for these applications the artificial neural network models out-performed the regression and discriminant analysis methods.

In another example Yoon et al [124] studied the capabilities and limitations of discriminant analysis and artificial neural networks for an application to predict stock price performance. An examination of the influence of each independent variable on the

output showed that the artificial neural network model was a good estimator. In evaluating the predictive power of the two methods, the back-propagation network outperformed the quadratic discriminant analysis model for this application [124].

Some of the background and theory, as well as some of the problems associated with artificial neural networks were discussed in this chapter, as well as some details of the structure of back-propagation and cascade networks. This prior research into these issues provided the necessary link between the problem of development effort estimation, and the tools which were applied and evaluated.

Chapter 4

Research Methodology

4.1 Introduction

In this chapter the broader details of the research design are discussed. The total project is composed of a series of six research projects using different datasets, each with their own set of trials. Each project is part of the overall research design. The general methodology and the integration of the various trials into the overall research design are discussed in Section 4.2. As each research project is reasonably autonomous and the research methodology details are slightly different for each case, the detailed design issues are discussed in the respective sections in Chapter 5.

Other research design issues regarding model performance measurement and the neural network default parameter settings are discussed in Section 4.3. Section 4.4 describes the research population, and Section 4.5 the various tools which were used in the research project.

Before the main research project was designed and implemented a pilot study was conducted to assess the feasibility of the proposed use of artificial neural networks for development effort estimation. The design, implementation and analysis of the pilot study are discussed in Section 4.6.

4.2 General Research Design

The software development environment is complex and to estimate development effort accurately is a non-trivial task. Research results evaluating estimation tools have been disappointing [109]. Chapter 2 reviewed some of the estimation problems. Artificial neural networks have been developed successfully for numerous applications. In Chapter 3 some of these were discussed, as well as a review of some aspects of artificial neural networks which indicated that they may have the potential to be used for software development effort estimation.

To provide this research project with a coherent structure several different components were identified and organised to best achieve the research objective as stated in Chapter 1. The general research design involved the following eight steps:

- review of significant published metrics and neural network research
- development of a pilot study
- collection of suitable empirical project data
- establishment of neural network models
- training of suitable artificial neural networks on the empirical data
- testing these models on test or validation data to estimate the project development effort
- analysis of the prediction error and comparison of results
- development of an effort estimation software system using artificial neural networks

The pilot study revealed that suitable neural network models will require a reasonable number of weighted connections to enable them to capture the complex project development domain. The selected back-propagation network from this project had a 23-4-1 topology, which resulted in 96 weighted connections. Baum and Haussler [10] have determined that large networks require large numbers of observations to enable them to adequately learn the problem domain and capture this into their weight space

to enable good generalisation and accurate estimates to be made (see Section 3.5). As cascade networks are developed dynamically during training it is not possible to pre-determine the number of weighted connections that will be deployed in the optimum network.

The pilot study network topology would have required approximately 1000 training observations to fulfil the Baum and Haussler requirement. Considerable effort was made to acquire a suitable dataset for this thesis. The significant prior research (Chapter 2) showed that metrics research studies are conducted on data with considerably fewer observations, despite the shortcomings [75]. These datasets typically consist only of 15–100 projects [2] [7] [9] [31] [49] [50] [53] [59] [60] [64] [75] [78], and this places a restriction on their statistical analysis and the interpretation of results.

This neural network research project requires a large software development dataset in which many attributes have been recorded. Several datasets are available but typically these are too small to provide sufficient observations to fulfil the requirement discussed above. The availability of a suitable dataset is further restricted by very few development projects having reliable records of the numerous development attributes which have to be included in the model for accurate estimation.

In Chapter 2 some of the available datasets were reviewed, and none of these meet the requirements as set out above. A concerted effort was made to search for the availability of a suitable dataset. The only dataset which may have been suitable is owned by Software Productivity Research Inc., who claim to have the largest collection of software development information in the world [55] with approximately 6,000 projects in their database. Negotiations with Jones [55] to secure the rights to use some of this data failed when Software Productivity Research Inc considered the use of neural networks a threat to derive their proprietary metrics from this data.

A survey conducted by the Australian Software Metrics Association (Queensland) [3] indicated that very few companies in Queensland have reliable historical development data available. The feasibility of this author developing a new dataset by collecting project information was considered, but indications were that this would have been prohibitively expensive, and would have taken too long to accomplish. Research

publications [59] as well as discussions with a professional function point counter [77] suggest that the time required to reliably count one 3,000 function point system will take approximately one week. Even if the size of projects in the proposed dataset was restricted to projects with an average size of approximately 1000 function points, it would still not have been feasible to collect the minimum amount of software development data to allow complete analysis using neural networks.

The possibility of developing an automatic function point counting tool for the 4GL (Fourth Generation Language) environment was investigated and proposed by Wittig and Finnie [121]. To automate function point counting so that it is reliable and widely accepted is a non-trivial task [87] and would have resulted in a separate research project, so that it was not considered to be feasible as a means of collecting data for this research project.

The difficulty of collecting project data is reflected by metrics organisations in the United States, United Kingdom, and other European countries working for many years to collecting software development data with little success [83]. The Australian Software Metrics Association (ASMA) project data (Release-5) [4] contains information on 136 projects, and has been widely accepted. Many international organisations will in future be contributing their project data to this dataset [83]. The cost and effort of collecting the ASMA dataset is spread across the many contributing organisations, and has the organisational backing and support of the ASMA organisation to coordinate and standardise many of these activities.

It would not have been possible for this author within the restrictions of a thesis research project to collect sufficient data to satisfy the neural network requirements for an adequate analysis of effort estimation accuracy. An alternative approach to one large single dataset was taken to overcome this problem. To evaluate the potential of neural networks it was considered important that they be assessed across a wide spectrum of issues which cover various aspects of software development, and artificial neural networks. To test them on only a relatively small single dataset may indicate good results on particularly that set under which those projects were developed, but for their wider evaluation, analysis on numerous data sets was considered necessary. Also as the number of observations will generally be limited in most empirical

situations, it is considered necessary to test the neural network performance under these imperfect conditions.

The following set of six research projects, each with numerous trials, were combined to achieve the overall thesis objective. Hornik et al [46] claim that neural networks are universal approximators which are capable of estimating any measurable function. The first project is designed to test and demonstrate neural networks' capability of extracting and capturing in their weight space a known productivity function (Section 5.2). In the second project the SPQR/20 metrics tool is used to generate a large dataset of simulated project data. This is then used to train the neural networks and assess the prediction accuracy and generalisation ability on test project data.

In Project 3 a project database of 81 observations is used to develop neural networks to enable the measurement of the effort estimation error. The limitation of this dataset, apart from the limited number of observations, is that very few of the significant development attribute factors had been recorded. This will afford the neural networks only a limited amount of relevant information from which to learn. This project will test the neural network capability under such restricted conditions.

Kemerer [60] in his study had assessed the estimation capability of numerous models. That very limited dataset is used to train neural networks on a rolling rotational basis in Project 4. The neural network estimation errors are then compared to the model estimates in the above study. Even though the models were developed under different conditions, the comparisons should be useful.

In Project 5 various trials were conducted on a dataset of 28 projects, which in addition to a size measure of all projects, includes data on the 14 technical complexity adjustment factors, as well as 21 cost drivers. Typically project datasets in the commercial sector are small. Projects 3–5 were designed to assess the estimation capability of neural networks under such imperfect conditions, as the 'ideal' dataset is not always available. If the management choice has to be made, due to the unavailability of perfect data, between an imperfect estimate from the neural network, or no estimate at all, the former may be preferred. These experiments were designed to test the neural network estimation models operating under such restrictions.

The final research project was conducted using the ASMA Release 5 project database [4]. Even though a greater number of observations would have been preferred, the number of projects is substantially more than the other empirical datasets. Various parameter settings, data scaling techniques, size measures and inputs are tested to ensure the selection of the correct technique for estimation error minimisation in this and the above research projects.

In this research project a comparison of the neural network estimates was also made to those generated by the k-nearest neighbour approach [82], as well as a comparison of the cascade and back-propagation neural network effort estimates

4.3 Specific Procedures

Some specific procedures applicable to the overall research design and some of the factors leading up to this design are discussed here. The details of some procedures which are specific to particular experiments are discussed in the respective sections describing these.

4.3.1.1 Pilot study

A pilot study was conducted to test the estimation accuracy of back-propagation networks on a set of development data. The dataset included projects across a large size range, and with typically large variations in productivity [123]. The pilot study included gaining experience on the effect of various back-propagation parameter settings, as well as various network topologies with varying numbers of hidden layers. A review of the pilot study is given in this chapter in Section 4.6.

4.3.1.2 Research design

The general research design was discussed in Section 4.2. The project consisted of six research projects, based on six sets of research data, each of which form part of the overall research design framework. As each project is slightly different, the specific procedures have been included with the respective sections, including those of the pilot study.

4.3.1.3 Estimation Error Measurement

One of the objectives of this thesis is to assess the estimation error. Different error measurements have been used by various researchers [16], but for this project the main measure of model performance is the Mean Absolute Relative Error (MARE). MARE is the preferred error measure of leading software measurement researchers and is calculated as follows [50]:

$$\text{MARE} = \left(\sum_{i=1}^{n} \left| \frac{estimate - actual}{actual} \right| \right) \div n$$

where:

estimate is the network output for each observation

n is the number of observations

To establish whether models are biased and tend to over or under estimate, the Mean Relative Error is calculated as follows [50]:

$$\text{MRE} = \left(\sum_{i=1}^{n} \frac{estimate - actual}{actual} \right) \div n$$

Cascade2 calculates the Root Mean Square Error (RMSE) of a network training and test data at the conclusion of its training. As different transformations had been conducted on the goal values, the RMSE of the various networks are not directly comparable. In each case the test values were extracted from the network output file, converted back to their original form, and the MRE and MARE were then calculated.

4.3.1.4 Parameter Settings

In Chapter 3 the significance of the network parameter settings was discussed. For the various trials, for some parameters settings the default values of Cascade2 were accepted. Those default parameter settings which were never changed throughout the trials are listed in Table 4.1. The parameter settings which were changed for various trials are listed in the relevant sections in Chapter 5.

Table 4.1 Default Parameter Coefficients

Parameter	Value
Output Mu	2.00
Output Epsilon	0.35
Output decay	0.0001
Output change threshold	0.01
Output patience	8
Candidate input Mu	2.00
Candidate output Mu	2.00
Candidate input epsilon	1.00
Candidate output epsilon	1.00
Candidate input decay	0
Candidate output decay	0
Candidate change threshold	0.01
Candidate patience	12

Output Mu is used by Quickprop, the Cascade2 learning algorithm, to train the output weights. The output epsilon controls the amount of linear gradient descent to be used in updating the output weights and is similar to the learning rate setting in back-propagation networks. The output decay coefficient times the current weight is added to the slope at the start of each output epoch, and keeps the weights from growing too big. This is similar to the momentum setting in back-propagation networks.

The output change threshold parameter means that the error must change by at least this fraction of its old value to count as a significant change. The output patience determines the number of epochs for which if there is no significant change in the error, the network stops training.

As with the output units, the candidate input Mu is used by Quickprop to train the input weights to the candidates in the pool. The candidate output Mu is used by Quickprop to train the output weights of the candidates. The candidate input epsilon controls the amount of linear gradient descent to use in updating candidate input

weights. Similarly the candidate output epsilon factor controls the amount of linear gradient descent to use in updating candidate output weights.

The candidate input decay factor times the current candidate input weight is added to the slope at the start of each candidate training epoch, and keeps these weights from growing too big. The candidate output decay coefficient performs a similar function for the candidate output weights. The candidate change threshold coefficient determines the threshold level of change in the error that is considered significant. The candidate patience determines the number of epochs the network will run without significant change in the error before the network stops.

4.4 Research population

The overall research design is composed of six sets of projects based on six research datasets. It was decided to keep each project reasonably autonomous, with each section describing procedures specific to that particular set of trials. Similarly a description of the individual datasets has been included with the relevant sections in Chapter 5, together with the analysis and results.

For Research Projects 1 and 2 simulated data was used, while for all the other experiments actual project data was used.

4.5 Tools used

4.5.1 Software

The main tool emphasis of this project has been on the use of artificial neural networks. The Neuralyst version 1.3 back-propagation network from Epic Systems was used for the pilot study and for some comparison trials. Neuralyst runs under Microsoft Excel and this author developed some macros to assist in the automatic recording of output results.

Most of the research was conducted using Cascade2 version 1.03. This is a cascade network, and is a derivative of cascade correlation networks. This software was

developed by Fahlman [25] and is not yet available for general release and user documentation is not available. Some minor improvements were made to the software by Queensland University of Technology [21], and these were incorporated in the software version which was used.

The network input and parameter setting is achieved through a network text file. To develop this the data transformations and scaling were done on Microsoft Excel 5.0. This data was then copied as unformatted text into Microsoft Word 6.0, where it was adapted into a form suitable for Cascade2. The net file header was added at this stage.

The output of Cascade2 was first imported into Word 6.0, where it was transformed into a form suitable for analysis on Excel 5.0. This research project benefited from some of the useful features of the above tools which eased the development of the numerous network files as well as the analysis of results.

For the generation of the simulated project data SPQR/20 from Software Productivity Research Inc was used. The data for the manual input into SPQR/20 had been prepared using Excel 5.0.

4.5.2 Hardware

Cascade2 was run on a Sun SPARC station LX, with 24Mb RAM, running under the operating system Solaris version 2.3.

Microsoft Word 6.0 and Excel 5.0 were run on a Commodore 486-33c, under Windows 3.1, and on an Apple Macintosh SE/30, using the System Z-7.0 operating system. Both these machines were also used as terminals to access the Sun workstation.

4.6 Pilot Study

4.6.1 Introduction

When the thesis research was started a literature search was conducted. This was not successful in locating any research publication concerned specifically with the use of

artificial neural networks for software development effort estimation. Consequently it was decided to conduct a pilot study [120] to examine the feasibility of using back-propagation neural networks for this purpose. The objective was to examine on a small project dataset the behaviour of the neural network and to obtain some indication of the likely prediction accuracy.

This section reviews the research methodology that was used, and the analysis of the results. As there were no previous studies to guide numerous neural network issues specifically related to effort estimation, such as scaling techniques and parameter settings, it necessitated drawing on previous neural network experience from other problem domains and applying these. In addition the pilot model was refined through trial and error. As it was a pilot study it was not tested comprehensively. The study was terminated when it was considered to have met its objectives.

4.6.2 Research Methodology

4.6.2.1 Data Collection

For this study development data from 15 commercial systems, developed by information systems professionals, was recorded [119]. The size of the systems was measured in unadjusted function points as it appears to be a more consistent measure than source lines of code [50] [60] [72].

Development effort was measured in development hours. All activities, starting from the specification stage and through to that stage where the product is ready to be delivered to the end-user are included in the development time. The time spent on documentation is therefore also included. Excluded is the time required for the formal user acceptance tests, as well as end-user training. The definition of a development hour is the actual time spent on the project, and also includes all time spent on attending meetings directly related to the software development, but excludes times such as public holidays, leave, illness, and development staff training.

The development attributes which were included in the study were selected after reviewing some previously published research [2] [12] [16] [49] [57] [114]. The selection was refined by interviewing the information systems managers of 10

commercial organisations to establish which factors they, with their development experience, considered to have a significant impact on development productivity.

Precise descriptions of the various development attributes were developed, to attempt to ensure data consistency across the dataset [119]. No inter-rater reliability study was conducted [59], as some of the systems were very large and this would have been costly. In the determination of the average size of the development team both analysts and programmers were included in the calculation, as well as project managers and program librarians directly involved with the development.

4.6.2.2 Research Data

The systems ranged in size from a small 29 function point system to a system of 4669 function points. If uncommented 4GL source lines of code (SLOC) is used as a size measure, the systems ranged in size from 600 to 571,000 SLOC. The development effort required to develop these systems ranged from 40 to 81,270 hours. A system of less than 300 function points would be considered a small system, while a medium system would be between 300 and 800 function points, a large system between 800 and 1000 function points, and a very large system would be greater than 1000 function points [23]. A summary of these details is given in Table 4.2.

The size unit of unadjusted function points (UFP) was used. There has been some criticism regarding the choice and weighting of the function point adjustment factors [56] [63] [103], and as it is easy to include these into the neural network model, the technical complexity adjustment to generate adjusted function points was not made. For this study software development productivity is defined in the economic sense [57], and is expressed as the amount of output produced per unit of input, and the unit used was unadjusted function points per development hour.

An examination of the productivity range indicates that this varies from 0.05 to 2.14 function points per hour. Data with such a large size and productivity range, with significantly different development attributes, and which inherently contains a lot of noise, further complicates effort estimation.

Table 4.2 Pilot Study Data Set

4GL System	Dev Hours	Lines Code	Size UFP	Productivity UFP/Hr
1	5 027	88 325	1 842	0.37
2	1 680	44 204	905	0.54
3	13 300	170 557	4 191	0.32
4	98	1 535	208	2.12
5	588	2 019	342	0.58
6	450	1 600	164	0.36
7	40	1 400	29	0.73
8	81 270	298 843	4 113	0.05
9	35 000	562 500	3 486	0.10
10	240	600	286	1.19
11	100	3 241	214	2.14
12	4 864	25 000	2 758	0.57
13	1 200	30 000	1 913	1.59
14	2 500	571 000	4 669	1.87
15	1 400		850	0.61

To convert the size of a system measured in function points to a development effort estimate, a productivity factor has to be applied. Productivity varies not only by programmer, but also by project size [23]. This implies that the effect of project size on productivity has to be quantified to enable an accurate estimate of development effort to be made [32]. With a limited sample size multiple regression analysis proved difficult to derive accurate effort estimates, and attempts have been made using the Analytic Hierarchy Process [33] to prioritise the effect of the various development factors.

4.6.2.3 Neural Network Model

For this study, back-propagation artificial neural network models were used. Back-propagation networks are the most generalised neural networks currently in use [79] and this approach was chosen in preference to Hopfield and Kohonen networks. Nelson and Illingworth [79] discuss some of the many networks which have been developed and give guidelines for possible applications. As software development

estimation is not a time series problem, approaches such as finite impulse response (FIR) and recurrent networks were not considered.

The back-propagation network requires data from which to learn. To learn the network calculates the error, which is the difference between the desired response and the actual response, and a portion of it is propagated backward through the network. At each neuron in the network the error is used to adjust weights and threshold values of the neuron, so that at the next epoch the error in the network response will be less for the same inputs. This corrective procedure is called back-propagation and is applied continuously for each set of inputs or training data. The training data should consist of as much relevant data as possible. In practice one does not usually have the luxury of a perfect training data set. With limited data one has to consider the trade-off between having as large a training set as possible and still leaving sufficient data points to test and validate the model.

For this project the data were divided into three sets. The training set comprised ten developments, the test set three, and the validation set two. The data for each category were randomly chosen, except that the data in the test and validation sets was not allowed to be larger or smaller than the largest and smallest developments respectively in the training set. This was done so that predictions were not made outside the data range on which the network had been trained.

The inputs were unadjusted function points, average development team size, systems analyst capability, systems analyst experience, the level of requirements volatility, the level of processing complexity, whether reusable code was developed or not, the required processing reliability, and the capability and the experience of the programmer team. All the inputs except for the average development team size and the unadjusted function points reflecting the system size, were converted into a binary notation. The target against which the network was trained was the development hours of the systems in the training set. The accuracy of the development effort estimate was taken as the Root Mean Square Error (RMSE), proportionate to the size range of the systems in the data set.

To avoid over-training the network, the dangers of which are discussed later, and to be able to monitor the generalisation capability of the network, the training error and the

prediction error were recorded at 100 epoch intervals. As the initial randomly generated starting network weights affect network performance these were saved, and the network was then re-run using the same starting weights, and stopped when it had reached its minimum prediction error.

4.6.3 Analysis and Results

Network models were developed with various combinations of inputs selected from the attributes mentioned above. The results were not particularly encouraging and the prediction errors were erratic and not satisfactory. An examination of the results showed that the network appeared to consistently over-estimate the size of the very small systems, as well as to consistently under-estimate the size of the very large systems. For the remaining systems, the estimated development effort was more accurate.

4.6.3.1 Network parameters

To try and improve the network performance, the learning rate and momentum were varied, as was the network architecture. Models with one through to six hidden layers were developed. Consistently the models with just a single hidden layer performed better, while the models with multiple hidden layers in many instances did not converge. Various activation functions were tried, and the popular sigmoid function consistently gave the best results.

To solve the problem of the network not training and predicting well on such a large target range, the natural logarithm of the development hours was used. This compressed the range and improved the network performance.

There is no clearly defined theory which allows for the calculation of the ideal parameter settings and as a rule even slight parameter changes can cause major variations in the behaviour of almost all networks [99]. It is through a process of trial and error and experience that settings are selected which result in a reduced average prediction error. The settings of the learning rate and momentum control the way in which the error is used to correct the weights in the neural network for each training case. When the learning rate is set to high values (close to 1) there is the possibility of

unstable behaviour, as evidenced by widely varying average error values. When the learning rate is set lower, the possibility of unstable behaviour is reduced, but training times are increased and there is a greater probability of getting stuck in local error minima. The higher the momentum, the larger the percentage of previous errors that is applied to the weight adjustment in each training case. For example, when the momentum is set at 0.5, then 50 percent of the weight adjustment will be due to the current error and 50 percent will be from the weight adjustment applied in the previous case.

Table 4.3 Training and Prediction Errors

Training Results		Prediction Results	
Iterations	Average Error	Iterations	Average Error
100	0.2748	100	0.1548
200	0.2060	200	0.1429
300	0.1504	300	0.1422
400	0.1210	400	0.1413
500	0.1063	500	0.1374
600	0.0971	600	0.1307
700	0.0892	700	0.1205
800	0.0820	800	0.1091
900	0.0757	900	0.0967
1000	0.0709	1000	0.0865
1100	0.0673	1100	0.0788
1200	0.0646	1200	0.0692
1300	0.0626	1300	0.0589
1400	0.0609	1400	0.0486
1500	0.0598	1500	0.0386
1600	0.0585	1600	0.0315
1700	0.0576	1700	0.0238
1800	0.0569	1800	0.0168
1900	0.0562	1900	0.0129
2000	0.0556	2000	0.0126
2100	0.0551	2100	0.0136
2200	0.0547	2200	0.0163
2300	0.0543	2300	0.0170
2400	0.0538	2400	0.0195
2500	0.0535	2500	0.0235

For this set of data a learning rate of 0.1 and a momentum of 0.7 gave good results. A neural network architecture of a single hidden layer using a sigmoid activation function tended to result in the lowest average prediction error. The best results were obtained with a 23-4-1 architecture. Table 4.3 shows the results up to 2500 epochs. The average training error is reduced steadily as the network trains, as is the prediction error. For this network the lowest average prediction error was obtained at about 2000 iterations. With further training the training error is further reduced, but the network does not generalise well, and from this point the average prediction error increases. The reason for this is that the network tends to curve-fit the training data, giving a low average training error, but this then leads to poor generalisation.

4.6.3.2 Network performance

A larger data set would have been preferable, but this type of information is difficult to gather. Other studies such as Kemerer [60] (15 cases) and [59] (27 cases) are also limited by data availability. A result of a small data set was that the network performance was significantly influenced by the initialisation weights which are randomly generated. Some sets of initialisation weights resulted in better convergence and a reduced average prediction error. As there is currently no known theory on the allocation of starting weights to optimise the network performance, and these are generated and allocated randomly, it meant that from repeated trials the weight set which resulted in the lowest average prediction error was selected.

The results of the network which gave the lowest average prediction error are shown in Table 4.4. The output has been normalised by converting it back from the natural logarithm and then rounding it. Both the actual system size and that predicted by the artificial neural network are shown. The first ten data sets comprise the training set. The next three comprise the test set, and the final two the validation set.

The error in the training set is not important. As mentioned above, by training the network further, this error could be further reduced, but this would have degraded the model's generalisation ability, resulting in an increased prediction error.

Examining the results in Table 4.4 reveals that neither the very smallest, nor the largest systems were included in the testing and validation set, as this would have meant that

they would have been excluded from the training set, and would have resulted in having to predict outside this range. Training was stopped when the prediction error was at its lowest. The prediction capability of a network is judged by the test and validation sets. The three systems in the test set took 100, 1200 and 4864 person-hours to develop. The model predicted 89, 1184 and 4429 for the three systems respectively, resulting in error predictions of 11.1%, 1.3% and 9.0%.

Table 4.4 Prediction Results

4GL System	Actual Development Hours	Estimated Development Hours	Percentage Error	Category
1	5 027	5 015	-0.24%	Training
2	1 680	1 652	-1.68%	Training
3	13 300	13 085	-1.62%	Training
4	98	269	174.41%	Training
5	588	377	-35.89%	Training
6	450	262	-41.69%	Training
7	40	62	55.42%	Training
8	81 270	62 236	-23.42%	Training
9	35 000	34 832	-0.48%	Training
10	240	240	0.11%	Training
11	100	89	-11.08%	Test
12	4 864	4 429	-8.95%	Test
13	1 200	1 184	-1.30%	Test
14	2 500	2 144	-14.24%	Validation
15	1 400	1 396	-0.27%	Validation

The two systems in the validation set required 1400 and 2500 person hours to develop. The prediction for the smaller system was 1396 person-hours, which is within 1% of the actual development time. For the other system the error was 14.2%.

4.6.4 Discussion

Within the limited dataset, back-propagation artificial neural networks appear to indicate the potential to be developed into good software size estimation models. In examining the model's performance the following factors need to be considered:

1. These systems were developed in an uncontrolled and natural (not artificial) environment. Only the data was gathered and there was no control over the development environment.

2. The software development environment is complex, with many and often interrelated factors affecting development effort. Currently available size estimation models have on occasions not performed well when applied to systems which were developed outside of a strictly controlled environment [31] [60] [78].

3. The range of the system sizes is large, as is the variation in productivity. Despite these difficulties the models performed well in terms of current level of prediction accuracy [31] [60] [78].

4. The research experience with the small data set highlighted the importance of the initial weights allocated to the network weight space. This makes the model development more difficult. Additional data, which is difficult to obtain in large sets, should reduce the influence of the starting weights and make training networks more stable [69].

5. The model is not difficult to develop and has the flexibility of being able to incorporate additional attributes as input if special circumstances warrant their inclusion.

The model has produced good results, as noted above, in being able to predict on average within 10 percent of the actual software development effort in the data set to which it was applied. It was thus concluded that the proposed research project to use artificial neural networks was feasible and warranted further research effort.

4.7 Conclusion

The general research design is influenced by the availability of reliable project data. Available datasets contain relatively few projects. The available datasets did not generally apply common standards and definitions to enable the combination of several datasets, to create a larger sample. The international recognition which the Australian

Software Metrics Association project dataset is receiving may within a few years result in a project database containing several hundred samples, using relatively consistent definitions and standards [83].

At present if neural networks are to play a useful role in project management they will have to be used on currently available project data. The pilot study indicated that neural networks may be able to estimate effort relatively accurately on currently available datasets. To ensure that the research results are not influenced by a chance result of a few observations, the research methodology encompassed four different project datasets, as well as simulated data to test performance on a large dataset.

This chapter considered the general research methodology and the roles each of the six research projects serve to achieve the overall research objective of assessing whether artificial neural networks are capable of accurately estimating software development effort.

Chapter 5

Analysis and Results

5.1 Introduction

In this chapter the analyses of the six research projects which were discussed in the general research methodology are described. Each project consists of several trials and the results of each are discussed in the respective sections. The discussion on the relevance of the research findings is reported in Chapter 7.

5.2 Equation Simulation

5.2.1 Introduction

Hornik et al [46] have established that multilayer feedforward networks are a class of universal approximators which are capable of approximating any measurable function from one finite dimensional space to another. This set of trials is conducted to establish whether the network architecture, topology, parameter settings, and data scaling used in some of the later trials is capable of extracting a measurable function from one finite dimensional space to another.

Finnie and Wittig [32] in the limited dataset which they had used had established the following productivity model:

$$P = 0.7703^{ats} \times 0.998^{fp/tm} \times FP^{0.168}$$

where

P	=	productivity in function points developed per hour
ats	=	average development team size
fp/tm	=	the number of function points developed per team member on the research project
FP	=	the system size measured in function points

This model is discussed in Chapter 2, and a graphical representation is shown in Figures 2.7 and 2.8.

5.2.2 Research Design

This set of trials used the above equation to generate input data to establish how well the neural network was able to extract the non-linear interrelated function from that dimension space into its own weight space. The cascade network Cascade2 was used for these trials. One thousand observations were generated for the training set. Only two inputs, average team size and system size (function points), were used. The output (network goal value) was calculated by transforming this input to produce the estimated developed productivity using the above equation.

For the cascade network, generally the default settings were accepted, except for the following:

A candidate pool of 20 was used. This was done to try and assure that sufficient 'good' candidates are generated each time. This slowed down network training, but as running time was not critical, it was considered a feasible trade-off. The maximum number of units permitted was set to 120, to try and assure that the goal is reached before this limitation restricts the network. Initial trials had indicated that the networks typically included approximately 50 hidden neurons, and thus the above setting should suffice for all cases.

The hidden unit type was set to sigmoid as the pilot study and other studies [108] had shown that the sigmoid activation function generally gave the best results. The output

type was set to linear as trials done by Diederich [21] had given good results for continuous output data. Various random initialisation weight range settings were used and the results are discussed later.

The error index is the normalised error function for continuous output training sets. The network stops training when the error index is less than the error index threshold. As accuracy is important, the error index threshold was set to 0.001.

The initialisation weights, both of the initial network, as well as all the candidate pools, affect network performance. Even a large candidate pool could not totally eliminate performance variations. For this reason each network was run five times, and the best result recorded for each run. Each network was given only two inputs, the system size in function points and the average team size, to approximate the above productivity model.

5.2.3 Research Data

The inputs were all randomly generated using the Microsoft Excel 5.0 random generator, in the range of 0–1, with an even distribution. The average team size input was then scaled using this randomly generated data to the range of 1–15, as this was also the range of the data in the Finnie and Wittig study [32] from which the equation was developed. The system size input was scaled arbitrarily in the range of 50 – 5000 function points as this was also approximately the range of system sizes used in the original study.

The data for the test set was developed to test the prediction accuracy against the range of data Finnie and Wittig used for the graphical representation of the model. Six system sizes, 100, 200, 400, 1000, 2000, and 3000 function point systems were therefore modelled for the test set. For each of the six sizes 15 observations were generated using average team sizes 1–15. This resulted in a test set of 90. The data used for the training set and the test set have not been included here as it is lengthy (20 pages) and similar data can be reproduced relatively easily.

5.2.4 Analysis and Results

For Trial 1 the above parameter settings were used, and the coding of the inputs and goal was done as follows. Hinton [44] proposed that the goal values should be scaled in the range of the sigmoid function, but that target values on the asymptotes tend to drive weights to infinity, slow down learning, and worsen generalisation. For this trial the target output, as well as the inputs were scaled in the range of 0.1 to 0.9.

For Trial 2 the inputs were scaled in the following manner. The average team size was divided by 10, and the function point size was divided by 1000. The development productivity was not transformed. The team size input ranged from 0.0721 to 1.5699, the function input from 0.0503 to 4.9945, while the goal values (productivity) ranged from 0.0002 to 1.3668.

In Trial 3 the data is scaled exactly as in Trial 2, except that now the goal values were scaled in the range of 0.1 to 0.9, to avoid any values on the asymptotes. This was done to assess this scaling scheme as suggested by Hinton [44] for the productivity data.

For Trial 4 the inputs were derived by calculating their natural logarithm. As in Trial 2, the development productivity was not transformed. The team size input ranged from 0.6725 to 3.7536, the function input from 3.9172 to 8.5161, while the goal values (productivity) ranged from 0.0002 to 1.3668. The input range was compressed by this transformation as trials conducted for the pilot study for this thesis suggested that the back-propagation network trained faster when this was done.

Table 5.1 Results of Trials 1–5

Trial No	MRE	MARE	Hidden Units	Epochs Trained
1	0.028	0.059	47	14,524
2	0.036	0.063	55	18,450
3	-0.014	0.064	35	9,907
4	0.009	0.036	60	22,266
5	-0.0089	0.051	60	16,942

In Trial 5 the data transformations are as those in Trial 4, except that the output values were transformed by calculating their natural logarithm values, which were then scaled in the range of 0.1 to 0.9.

The results of the five trials is shown in Table 5.1

5.2.5 Discussion

The prediction errors for Trials 1–3 were similar. This suggests that the linear scaling of the input data in the range of 0.1–0.9 did not reduce the prediction error, when compared to the scaling in Trials 2 and 3. Nor did the scaling of the output data, which in the normal form ranged from 0.0002 to 1.3668, to the 0.1–0.9 range reduce the prediction error.

The MARE of Trials 4 and 5 is slightly lower than the other three. As in the pilot study the compression of the input range using the natural logarithm transformation gave good results. The data scaling in Trial 4 gave the best results, with the lowest overall MARE of 0.036. As the range of output values is not large, using them in the untransformed state gave the best results, and there was no benefit of applying the natural logarithm transformation in this case. The prediction results are given in Appendix D, Table D.1.

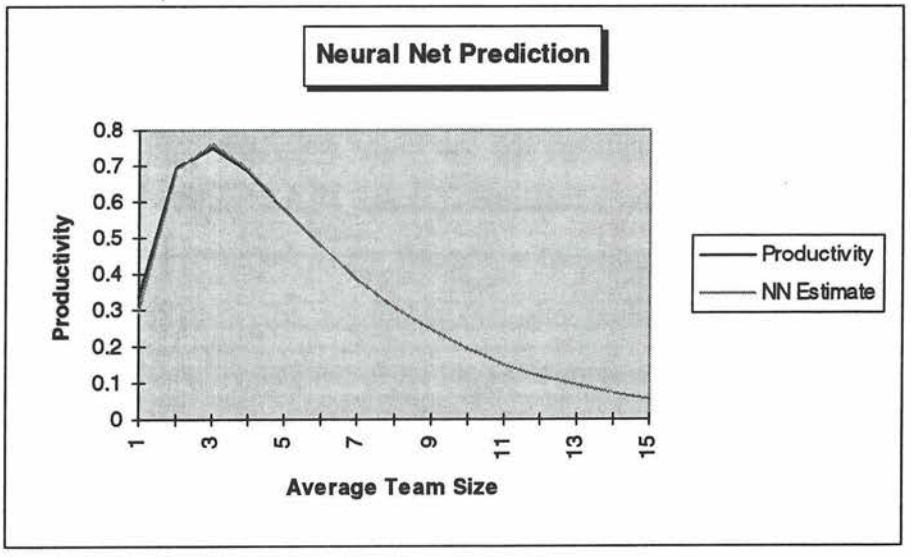


Figure 5.1 NN Prediction for 1000 FP System

As was mentioned previously, the MARE was 0.036, while the MRE was 0.009, which suggests virtually no estimation bias to either over or under estimate. The results are further illustrated by charting the actual and the estimated productivity for a 1000 function point system. This is shown in Figure 5.1, which indicates a good fit.

The data in Appendix D Table D.1 was analysed to determined whether the predicted productivity is significantly different from the calculated productivity. To test this, the null hypothesis that the mean difference between paired values is equal to zero (H_0: $\mu = 0$) is established. The t-Test for paired sample means was used to test this hypothesis. The results are shown in Table 5.2

Table 5.2 t-Test: Paired Two Sample for Means

	Equation	Estimate
Mean	0.3438	0.3437
Variance	0.1029	0.1019
Pearson Correlation	0.9993	
Hypothesised Mean Difference	0	
t Stat	0.0777	
P(T<=t) two-tail	0.9382	
t Critical two-tail	1.9870	

The computed t-statistic is 0.0777 which is well below that of the critical two-tail t value of 1.9870, and the null hypothesis is thus not rejected. This indicates that the paired productivity and predicted values are not significantly different. Also noted is the high probability of almost 0.94 that the computed t value would be less or equal to the critical t value, and the very high Pearson correlation coefficient of 0.9993. Both these statistics strengthen the argument that the prediction values of the neural network is not significantly different from the actual productivity values.

5.2.6 Conclusion

These trials have shown that neural networks are capable of successfully capturing the above mathematical relationship into their own weight space dimension. Unlike statistical techniques such as multiple regression, where a prediction is made with a certain probability confidence factor, neural networks have no such measure. They rely on the confidence a user acquires from their performance evaluation on the test set and validation set if this is available.

If the neural networks in these trials had not shown to be capable of approximating the productivity function from one finite dimensional space to another, the confidence with which they are perceived to be capable of successfully capturing any other relationships from other software development data would have been reduced.

5.3 SPQR/20 Simulation Data

5.3.1 Introduction

Neural networks are recognised for their ability to provide good results when dealing with problems where there are complex relationships between inputs and outputs [92]. The software development environment from which development effort estimates are generated, are characterised by complex relationships [7] between the development attributes and total development effort.

To assess the development effort estimation ability of neural networks a training set is required which is large enough to permit the network to sufficiently capture the problem domain characteristics to facilitate good generalisation. In Chapter 3 the number of observations which are required for a training set to achieve adequate capture of a problem domain in the neural network weight space was discussed. Baum and Haussler [10] suggested a formula to calculate the minimum number of required observations to achieve satisfactory generalisation. According to Hinton [44] in practice this approximates to the total number of weights divided by the allowed

fraction of errors on the test set. The error fraction is assumed to be less than 0.125. This implies that as a guideline approximately ten observations are required for each weight in the network [21].

5.3.2 Research Design

In this section the neural networks are assessed in a complex environment which include many development attributes. This requires a large software development dataset in which several attributes have been recorded. Several datasets are available but typically these are too small to provide sufficient observations to fulfil the requirement discussed above. The availability of a suitable dataset is further restricted by very few development projects having reliable records of the numerous development attributes which have to be included in the model.

In Chapter 2 some of the available datasets were reviewed, and none of these meet the requirements as set out above. A concerted effort was made to search for the availability of a suitable dataset. A survey conducted by the Australian Software Metrics Association (Queensland) [3] indicated that very few companies have reliable historical development data available. The feasibility study of developing a new dataset by collecting project information indicated that this would have been prohibitively expensive, and would have taken many years to accomplish.

It was not possible for this author within the restrictions of a thesis research project to collect sufficient data to satisfy the neural network requirements for an adequate analysis of effort estimation accuracy. To enable such an analysis to be conducted an alternative approach is taken by generating simulated software development project data. To do this the SPQR/20 software estimation tool is used. SPQR/20 has given relatively good results in estimating development effort in a locally calibrated environment [53], which enables it to generate a dataset which reasonably represents the software development environment.

5.3.3 Generation of Simulation Data

To generate the simulated project data all input values were randomly generated using the Microsoft Excel 5.0 random number generator. Input values were generated for 1,000 projects, and were then keyed in manually using SPQR/20 to generate the required project development effort.

The selection criteria were chosen in such a way to make the dataset represent the common business and commercial applications. SPQR/20 has many options and these could easily have been included into the neural network structure. It was considered though that including the full range of options was excessive and confusing and some of the following restrictions were imposed, which were not deemed to make the project data less representative of typical projects.

SPQR/20 has the ability to generate effort estimates for new programs, as well as enhancement and maintenance projects. For the simulated dataset only new program effort estimates were generated. For this dataset it was assumed that all projects were developed in some high level database language. The SPQR/20 tool has eight estimation options to select the desired project scope. The simulated dataset was restricted to only one option, which was for systems which consist of multiple components or programs.

Schedule time compression and level of required software reliability and quality affect development effort. SPQR/20 has eight options to reflect the project goals in relation to these factors. For the dataset that was generated this was restricted to only one option, and it was considered that the choice of an effort estimate of projects using normal average development team sizes, schedules and quality requirements was appropriate for all projects.

The default effort unit selected was work hours, as the ASMA project dataset [4] uses this measure. In all cases a 40 hour working week was accepted, with no overtime being worked. The project data being simulated also assumed that all projects were internal programs used at a single location. This assumes that normal management, some requirements costs, and normal defect removal with testing was done by the developers themselves. This is a class typical of internal management information

systems and information systems applications, such as finance and manufacturing systems that are created for use at a single location.

SPQR/20 has the ability to generate effort estimates for 14 project types. For the simulated dataset all projects were considered to be interactive database programs, as these are typical in the environment discussed above.

5.3.3.1 Environmental Inputs

SPQR/20 allows for eight project environment development attribute classes, each with a choice of five categories. For the simulated project data three of these were not varied. For all projects it was arbitrarily assumed that the office facilities were adequate, with on average three staff members per office. It was also assumed that all user documentation was provided by the programmers or the users, with fully automated graphics/text support. The last assumption made under the environmental inputs was that on no project was use made of reusable code. The use of varying amounts of reusable code, of varying quality, written in different languages, requiring varying degrees of modification was considered an unnecessarily complicating factor. It was considered that this restriction did not detract from making the simulated project dataset typical of many current developments.

The following are the five environmental attribute factors, each with five choices, each requiring different development effort. For each category a random value in the range of 1–5 was generated.

Project Novelty

- Conversion or repeat of a previous program
- Functional repeat, but some new features
- Even mixture of new and repeated features
- Novel program, but with some well understood features
- Novel program, of a type never before attempted

Requirements

- Program developers are also the program users
- Working model or prototype, plus clear requirements
- Fairly clear user requirements
- Ambiguous or incomplete user requirements
- Ambiguous, incomplete, and rapidly changing user requirements

Program Design

- Reusable designs and automated graphics/text design support
- New designs and fully automated graphics/text design support
- New designs and partially automated graphics/text design support
- New designs and little or no design automation
- Informal or hasty design with no automation

Response Time

- Response time does not affect this project
- Subsecond response time is the norm
- One to five second response time is the norm
- Five to ten second response time is the norm
- More than ten second response time is the norm

Staff Experience

- All experts in the type of program being developed
- Majority of experts, but some new hires or novices
- Even mixture of experts, new hires, or novices
- Majority of new hires or novices, with few experts
- All personnel are new to this type of program

5.3.3.2 Complexity Factors

SPQR/20 uses three different aspects of complexity. These are:

1. The complexity of the problem or algorithms to be coded
2. The complexity of the code itself
3. The complexity of the data which the program or system will utilise

Objective measures exist only for code complexity. There are no objective measures that capture the complexity of problems or the complexity of data, so a measure of subjectivity is required. SPQR/20 uses the following parameters for complexity.

New Code Logical Complexity

- Simple algorithms and simple calculations
- Majority of simple algorithms and calculations
- Algorithms and calculations of average complexity
- Some difficult algorithms or complex calculations
- Many difficult algorithms and complex calculations

Fractional inputs, such as say 3.5, are permitted to fine tune estimates, but were not used in the simulated data. The logical complexity is aimed at the difficulty of the problems for the development team to develop and code the results. A complexity of 1 or less indicates a very simple problem, with nothing other than minor summation and selection being involved. A complexity score of 1 to 2 implies generally easy calculations with one or two slightly difficult equations or formulae. A score of 2 to 3 indicates the presence of normal mathematical difficulties, with some complicated equations, while a score of 3 to 4 is associated with systems programming, telecommunications programming, and other areas that have many difficult problems hidden in the software. A complexity score of 4 to 5 is allocated to systems which involve substantially difficult problems, such as trajectory calculations, radar tracking interpretation, and circuit simulations.

New Code Complexity

- Non-procedural
- Well-structured, plus standard reusable modules
- Well-structured, with small modules and simple paths
- Fair structure, but some complex modules and paths
- Poor structure, with many complex modules and paths

This classification deals with the structural complexity of the source code. Again fractional values are acceptable. A score of 1 indicates the use of a very powerful fourth generation language. A response of 2 to 3 implies the use of a system which is commonly developed by the organisation, and contains many standard elements. A score of 3 to 4 means fairly close adherence to the structured programming conventions, while a score of 4 to 5 implies the use of inexperienced programmers or the use of a language such as BASIC which does not lend itself easily to being highly structured. A score of 5 would thus suggest 'spaghetti' code with many GOTO statements.

New Code Data Complexity

- Simple data with few variables and low complexity
- Several data elements, but simple data relationships
- Multiple files, data interactions, and file updates
- Complex file structures, data interactions, and updates
- Very complex data elements, interactions, and updates

This classification deals with the complexity of both the file structure and the data elements which the system utilises. Again decimal values are permitted. A score of 1 suggests that the program requires almost no data at all, such as a game or a simple spreadsheet. A response in the 1 to 3 range indicates a program which is not heavily dependent on large files or databases, and does not require much selection, validation, and update logic. A score range of 3 to 5 indicates programs that manipulate multiple files such as complex database applications with large numbers of fields and sophisticated selection and validation criteria.

For the simulated database there were thus eight classes of project attributes, each with five categories. This results in a combinatorial explosion of different combinations of choices, and in this case to exhaustively model all options would require 390,625 projects. In addition to this, each combination of project attributes combines with a project of varying size, which in each case affects the magnitude of the various factors differently.

From the simulated dataset of 1000 observations it is thus not possible to include all possible options in the training set. It was decided to arbitrarily place 900 observations into the training set. With such a small percentage (0.026 percent) of the total possible options being included in the training set, it is probable that the majority of the cases in the test set are not represented in the training set, and the neural network estimate accuracy will reflect its generalisation capability.

5.3.4 Network Input Coding

The various categories in the different classes do not represent some ordinal scaling and no presumptions are made in this regard. For this reason each category was considered a single input, and all were encoded in binary notation, giving them a value of 1, if that was the randomly generated selection, else 0. All the other classes which were restricted to one option in each case would have meant that there would be a repetition of 1s for all of these for all projects. It was decided to combine these into one single surrogate category to represent all of them as they were common to all projects. This then resulted in there being 41 binary inputs and one input for the systems size (function point) which was a continuous value. Similarly the single output, which was the development effort generated by SPQR/20 covers a continuous value.

The system size and development effort cover a large range and vary considerably and thus represented a scaling problem. Various trials were conducted to establish which method gave the lowest prediction error.

The data collected in the simulated project set also included the development effort, staffing level, and schedule for each of the seven development phases (planning, requirements, design, coding, integration/test, documentation, management). This is not included here, nor in the appendix as it is reasonably voluminous (approximately 25 pages) and the individual values are not crucial to the explanation of the research results. Also similar data may be reproduced to replicate these trials, as all selected inputs were randomly selected.

5.3.5 Research Data

The total number of projects in the dataset was 1,000. Some statistical details are given in Table 5.3. To note is the large range of system size (109–15,571 FPs) and development hours (2,162–912,309). Such size range was generated to fully test the neural networks prediction capability, but it did present itself with the input and output network scaling problem. Of interest is the productivity range which was generated by SPQR/20. Here the project with the highest productivity is approximately ten times that of the project with the lowest productivity. This range of development productivity is typical of commercial development productivity as was discussed in Chapter 2. This meant that the neural network had to accommodate large development attribute influences to generate an effort estimate from the function point size.

Table 5.3 Details of Simulated Project Dataset

	Function Points	Effort Hours	Schedule Months	Productivity FP/Hr
minimum	109	2,162	10.2	0.007
maximum	15,571	912,309	103.2	0.072
mean	2,340	116,238	39.4	0.025
median	2,080	96,772	38.8	0.023

5.3.6 Neural Network Parameters

All network parameters were set to the same values as for the networks used for the other simulated data in the equation simulation in Section 5.2. The system size (function points) network input, and development hours network output were also scaled using similar techniques as for those trials. Again as the network initialisation weights affect the prediction error, each network was trained five times, and the best one was then selected. In each case the selection criteria was the MARE of the test set.

For the scaling scheme which resulted in the lowest prediction error the function point input and the development hour output were transformed using the natural logarithm, and the result was then scaled in the range of 0.1 to 0.9. This means that the input and output ranges were finally compressed from 109–15,571 and 2,162–912,309 to 0.1–0.9.

5.3.7 Analysis and Results

The network with the lowest prediction error had a MRE of -0.003. This indicates that there was little bias in the estimates, with over and under estimate errors virtually cancelling each other out. The MARE was 0.045. The results for the 100 project test set are given in Appendix D, Table D.3. To facilitate an overview of a project and the neural network estimate, the network inputs are given in the table in their 'raw' form. The inputs 1 to 8 are the eight project attributes as discussed above, showing the randomly generated classification selection in the range of 1 to 5. The other input is the system size and the unit of measure is function points. The only output from the neural network in this case was the estimated development effort. The actual effort as calculated by SPQR/20 is shown, as well as the neural network estimate. In both cases the unit of measure is thousands of hours. The final column shows the absolute relative error (ARE) for each estimate.

To indicate the network performance a histogram of the AREs is shown in Figure 5.2. As there are 100 observations in the test set, the frequency on the Y-axis also reflects the frequency percentage. In the first column is the frequency of estimates for which the AREs was smaller or equal to 0.02. The other columns indicate the frequency for AREs equal or less than 0.04, 0.06, 0.08, and 0.10 respectively. The final column shows the frequency of AREs which are greater than 0.10.

The results indicate that 83 percent of the estimates were within eight percent of the actual value. To establish the statistical significance of the estimate a paired two sample for means t-Test was conducted. The null hypothesis was that the ARE between the project effort and the neural network estimate are equal to zero (H_0: $\mu = 0$). The results are shown in Table 5.4. The calculated t statistic is well below the

critical t value, and the hypothesis thus not rejected. This indicates that the neural network estimate is not statistically significantly different from the project effort.

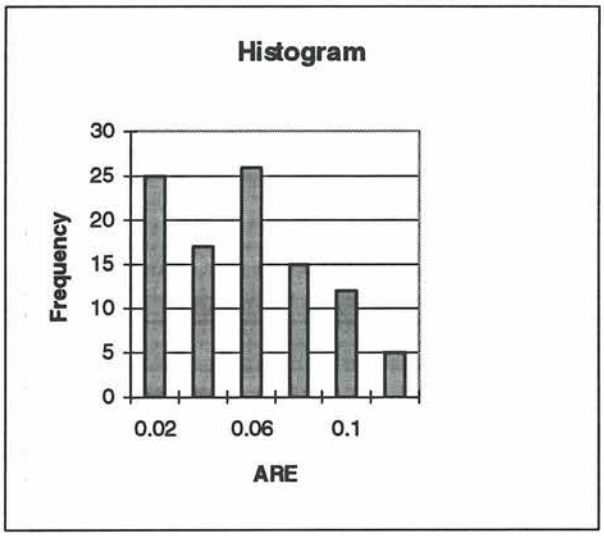


Figure 5.2 ARE Frequency in Test Set

Table 5.4 t-Test: Paired Two Sample for Means

	NN Estimate	SPQR/20 Effort
Mean	117.3	117.9
Variance	9066.5	9097.4
Pearson Correlation	0.995	
Hypothesised Mean Difference	0	
t Stat	-0.6108	
t Critical two-tail	1.9842	

5.3.8 Discussion

The dataset which was used had a large range of system sizes, total development effort, and development productivity. As was discussed in Chapter 2, to facilitate an accurate estimate of development effort, it is necessary to model the effect of the various development effort attributes, as the variation in system size alone is inadequate.

In the absence of a suitable large dataset of actual commercial projects a simulated project development dataset was generated to test the cascade networks ability to model the effect of various development attributes to enable it to accurately forecast development effort.

On the dataset which was used cascade networks have shown the ability to accurately estimate development effort which is not significantly different from the actual project effort. All project effort estimates were within 25 percent of the actual value, and the MARE for all 100 estimates was 0.045.

In many respects the simulated data is similar to actual project data. The one aspect which was not modelled was the addition of noise to the dataset, as project data generally contains noise. Some of the factors which introduce noise into project data were discussed in Chapter 2, and include among many others inter-rater inconsistencies, subjective metrics, and human error. Neural networks are known for their ability to provide good results even when input data is distorted by noise [92]. It will be necessary to show that neural networks have the ability to make accurate estimates at the noise levels which are typically found in project data, but this will only be possible when large databases of project data, such as the ASMA database, become available in the future.

If neural networks had not been able to estimate effort from the simulated project data, their ability to be able to do this from actual software development data would be questionable. The cascade network's ability to produce an effort estimate which is not statistically significantly different from the project data, suggests their potential as an added tool in the project manager's arsenal.

5.4 Desharnis Data

5.4.1 Introduction

Actual project data contains noise due to various factors such as inter-rater inconsistency, poor counting techniques, and human error, which complicate the

estimation task. Development effort is also affected by many development attributes. Jeffery et al [53] determined that in their study only approximately 40 percent of effort variation was as the result of system size. In such projects the use of only the function point size of a project as the only input into a neural network to estimate development effort will not yield accurate estimates, as the information is too limited and does not account for the effect of other factors.

The objectives of this set of trials is to assess:

- neural network estimation accuracy in restricted datasets, with limited input attributes
- scaling techniques and their effect on the effort estimation error
- the effect of some network parameter setting variations

5.4.2 Research Data

Project data from 81 projects is used [20]. The size of the systems is measured in function points and development effort in development hours. The minimum, maximum, mean and median values are shown in Table 5.5. The systems are of a relatively small size when compared to typical commercial software development projects [123]. In common with other commercial projects is the range of productivity, with the maximum productivity being 14 times that of the lowest [123].

Table 5.5 Desharnis Project Data Summary

	Unadjusted FPs	Function Points	Effort (Hours)	Productivity (FP/Hour)	Schedule (months)
Minimum	73	62	546	0.024	5.0
Maximum	1127	1116	23940	0.341	52.0
Mean	304	289	5034	0.081	27.6
Median	266	255	3647	0.058	28.0

Apart from the above project data, the number of years experience the project team had with the development methodology and tools, and the number of years experience of the project manager, as well as the development language environment, was the other data which had been recorded.

The inputs into the neural networks were

- team experience
- project manager experience
- development language
- unadjusted function points

The only output/goal was development effort (hours)

The team experience and project manager experience inputs were taken at their numerical values, and were then normalised in the range of 0–1, as was proposed by Hinton [44]. The language types, of which there were three, were encoded in binary format. For the initial trials the size measure was unadjusted function points, as they had not been subjected to any technical complexity adjustment which may have been inappropriate.

5.4.3 Research Design

For this research study cascade networks were used to estimate project development effort. The large range and relative magnitude of the system size and development effort created a problem for Cascade2. To resolve this three sets of scaling trials were conducted to determine which of these resulted in the lowest prediction error. In Trial 1, the function point inputs were normalised in the range of 0–1, and development hours outputs were ranged from 0.1–0.9. Hinton [44] suggests that target values be within the range of the sigmoid function, and thus the selected target output range of 0.1 to 0.9 was used.

In Trial 2 the inputs were the same as for Trial 1, except that function point inputs were scaled in the range of 0.07 to 1.13 by dividing the actual values by 1,000, and the target values (development hours) were scaled from 0.05 to 2.39 by dividing the actual hours by 10,000. The inputs and outputs for Trial 3 were the same as for Trial 1 except

that now for the function point inputs and the target development hours the natural logarithm was taken of the actual values. This was done to test the scaling method to establish whether a logarithmic scaling would result in a reduced prediction error. For this trial the function point inputs ranged from 4.29 to 7.03, and the development hours from 6.30 to 10.08.

The training set was made as large as possible, and was arbitrarily selected to be 71 observations. These were selected randomly, again using the Microsoft Excel 5.0 random generator. This left 10 randomly selected projects for the test set, to test the network generalisation. To avoid attempting to generalise outside the training set, the five largest and five smallest projects in the dataset were identified and had been excluded from the pool of available candidates for selection for the test set.

5.4.4 Analysis and Results

For each of the three trials the neural networks were run five times and the model with the lowest mean absolute relative error (MARE) for each trial was selected. The results are shown in Table 5.6.

Table 5.6 Results of Trials 1–3

Trial	Number of Hidden Units	Epochs Trained	MRE	MARE
1	24	6053	-0.118	0.515
2	12	3225	0.071	0.660
3	15	4315	0.473	0.667

Trial 1 gave the lowest MARE of 0.52, with Trials 2 and 3 giving similar error values of about 0.66. This result of an average prediction error of about 50 percent is not unexpected. The productivity variation, which determines the system size (function points) and development effort relationship, exceeds 1,400 percent in the dataset. Also many factors have a significant influence on the development effort. Desharnis [20] had

collected data on only four factors, which were used in the models, to explain the large productivity variation.

The Trial 1 model was considered the preferred network, and further trials were conducted to assess whether the prediction error could be reduced. From the dataset, as the development schedule is given, the average team size could be determined. This was considered a reasonable input for effort estimation as the team size is an *a priori* management decision. For Trial 4 the average team size was scaled in the range of 0.1 to 0.9. The network for Trial 5 is similar to Trial 4 except that a further input was included as a surrogate for all other factors, which were assumed to be common to all projects, and given a value of 1. Again five networks were trained for each model, and the best results of are shown in Table 5.7.

Table 5.7 Results of Trials 4 & 5

Trial	Number of Hidden Units	Epochs Trained	MRE	MARE
4	16	3984	0.140	0.355
5	12	2886	-0.064	0.520

Table 5.8 Prediction Results of Trial 4

Project No	Actual Effort	Estimated Effort	Error	ARE
19	4494	4476	-18	0.004
26	3164	4260	1096	0.346
74	595	891	296	0.498
70	1155	1236	81	0.070
37	1435	569	-866	0.603
79	9520	7769	-1751	0.184
30	3948	5532	1584	0.401
61	2926	3763	837	0.286
52	3136	2239	-897	0.286
47	4004	7517	3513	0.877
MARE				0.355

The models for Trial 5 were no improvement to those of Trial 1, but Trial 4 gave an improved result, with the MARE improving from 0.52 to 0.36. This result suggests that team size may have a significant effect on development productivity, which affects total development effort. The prediction results of Trial 4 are given in Table 5.8.

Further models from Trial 4 were trained with some network parameter changes. Two changes were assessed. One was to increase the network candidate pool to 100, to try and assure that the best possible candidate is generated and selected. The results did not improve from the models used for Trial 4, which suggests that for the dataset used the candidate pool of 20 was sufficiently large to ensure good candidate selection. A further trial was also conducted by setting the initialisation weight range to 0.1. Again this did not result in any reduction in the prediction error.

5.4.5 Discussion

The objectives for this research project were met. The neural networks converged and estimation error was determined with limited inputs. As expected the MARE of 0.36 was relatively high. Many more factors affect development effort, and it would have been unreasonable to expect neural networks to model their effect with no information on them.

The results of varying the scaling technique and some parameter settings suggests that the originally selected values were appropriate.

5.5 Kemerer Data

5.5.1 Introduction

In his study Kemerer [60] presents an empirical validation of the SLIM, COCOMO, Estimacs and Function Point algorithmic models. The objective of this section of the thesis research was to analyse how neural network model's prediction accuracy compares with the above models.

5.5.2 Research Design

Kemerer [60] in his study assessed the estimation accuracy of models on a database which was different from the data on which the different models had been developed. This was done to determine how well these models generalise outside the environment in which they were developed. To enable a comparison of a neural network model on a similar basis would have meant that it would have to be trained on some project data, and then tested on the data which Kemerer used for his assessment.

The research problem is in deciding which database to use to train the neural networks. The difficulty is that the end result will then depend on how closely that data resembles the data which Kemerer had used. Sections 5.2 and 5.3 have shown that neural networks have the ability to capture relevant relationships between inputs and outputs and the performance of the neural network will in a simple model where effort is estimated from system size only, be largely dependent on the characteristics of the training data.

To minimise this effect in the assessment a dual approach is taken. The first comparisons are made by developing networks on a rotational basis each time estimating one 'unseen' project using Kemerer's dataset. These estimates are then aggregated and used in the comparison. In the second stage neural networks are trained using two other databases, and these results are then compared to those trained on the Kemerer dataset for consistency.

5.5.3 Research Data

Kemerer's dataset comprises 15 projects [60]. Twelve of these were developed in COBOL. As the development environment may have a significant influence on productivity, and therefore on development effort, only the 12 COBOL projects were used for this analysis. As the original analysis of Kemerer had included only nine projects in the Estimacs assessment, the exclusion of the non-COBOL projects meant that in this case it was reduced to eight projects.

The inclusion of the development environment variable into the models was not feasible in such a limited dataset. For both the regression analysis and neural network

models a much larger dataset would have been preferable and would have enabled the development of more robust models. These trials are thus conducted within this limitation, but in the absence of other published comparisons this analysis is justified.

5.5.4 Estimation Models

5.5.4.1 Effort Estimate using Function Points

For this comparison, 12 networks were developed. In each case on a complete rotational basis, the training set consisted of 11 projects. These were used to train the network which was then used to predict the one 'unseen' project in the test set. Again as the initialisation weights influence the prediction accuracy, each network was trained five times. The MARE was calculated for each, and the network with the lowest MARE was then included in the comparison.

Table 5.9 shows the results of the trial. It shows the actual effort and the relevant SLIM, COCOMO, FPA, Estimacs, and Neural Network (function point based) estimates for comparison. The project numbers are those from Kemerer's study [60].

Table 5.9 Effort Estimates

Project No	Size FPs	Effort PM	SLIM Estimate	COCOMO Estimate	FPA Estimate	Estimacs Estimate	NN Estimate
1	1217	287	3858	933	344	230	299
2	507	83	100	151	92	111	70
3	2307	1107	11982	5819	731	524	697
4	789	87	2017	567	192	235	222
5	1338	336	3382	1316	387	688	257
6	421	84	263	312	62		32
8	993	130	1225	1206	265	389	90
9	1593	116	1454	4578	478		234
11	1611	259	1623	1576	484	624	228
12	789	231	513	584	192	325	91
13	691	157	3120	1124	157		151
14	1348	247	380	664	391		338

Table 5.10 summarises the MARE of the neural network, SLIM, COCOMO, FPA and Estimacs estimates for comparison. As only the COBOL projects are included in this analysis, the aggregate values differ slightly from those published by Kemerer [60]. As the Estimacs estimates were conducted on only a subset of the sample, the neural network comparison is repeated, using only the relevant projects.

The model which gave the lowest MARE in Kemerer's study was FPA (0.67). The neural network estimate resulted in an MARE of 0.45. Despite the reduction in MARE, the paired two sample for means t-Test indicates that the neural network estimate is not statistically significantly different. The small group size, with a large variance within each group contributes to this.

The variance of the MARE for various models is also given, and shows a similar trend to the MARE, again with the lowest variance being in the neural network estimate.

Table 5.10 MARE Comparison

Project No	SLIM Estimate	COCOMO Estimate	FPA Estimate	NN Estimate	Estimacs Estimate	NN Estimate
1	12.44	2.25	0.20	0.04	0.20	0.04
2	0.21	0.83	0.12	0.15	0.35	0.15
3	9.82	4.26	0.34	0.37	0.53	0.37
4	22.21	5.52	1.21	1.56	1.70	1.56
5	9.06	2.91	0.15	0.23	1.05	0.23
6	2.13	2.72	0.27	0.62		
8	8.40	8.26	1.03	0.31	1.99	0.31
9	11.54	38.46	3.12	1.02		
11	5.27	5.09	0.87	0.12	1.41	0.12
12	1.23	1.53	0.17	0.60	0.41	0.60
13	18.87	6.16	0.00	0.04		
14	0.54	1.69	0.58	0.37		
MARE	8.48	6.64	0.67	0.45	0.95	0.42
Variance	50.77	105.25	0.75	0.20	0.47	0.24

To assess whether the neural network training conducted on the Kemerer dataset gave the neural networks an advantage over the other models, two other neural networks were trained. The one was trained on the Desharnis dataset (Section 5.4) and the other on the Mermaid dataset (Section 5.6). In each case the test set was the Kemerer

dataset used in this section. The results are shown in Table 5.11. For simplicity only the FPA results are shown here, as the MARE is much less than the SLIM and COCOMO models.

Table 5.11 Consistency Comparison With Other Training Datasets

Model	MARE	Variance
FPA (Kemerer Study)	0.67	0.75
NN (Kemerer dataset)	0.45	0.20
NN (Desharnis dataset)	0.51	0.05
NN (Mermaid dataset)	0.45	0.07

The results indicate that the MARE of the neural network trained on Kemerer dataset in such a manner that it on a rotational basis each time estimates an 'unseen' test project, is consistent with the MAREs of the two neural networks trained on the Desharnis and Mermaid datasets. In both of these neural networks, as with the original network discussed above, the MARE was lower than that of the FPA model but the difference was not statistically significant.

5.5.5 Effort Estimate using Lines of Code

In Kemerer's study [60] the SLIM and COCOMO models use lines of code as their size measure and basic input. A similar set of trials to the above with 12 networks was conducted, with the input into the neural network being lines of code rather than function points, to test whether this measure of size reduces the MARE. The results are given in Table 5.12

The results show that the MARE is not very different and there is no statistically significant difference whether lines of code or function points are used as the size measure and as input into the neural network. This result substantiates Jeffery and Low's finding [50], where in their study, the prediction error between the two approaches was also not significantly different.

Table 5.12 Lines of Code Result

Project No	Effort PM	NN Est (FP)	NN Est (LOC)	MARE (FP)	MARE (LOC)
1	287	299	259	0.041	0.097
2	83	70	83	0.146	0.008
3	1107	697	355	0.371	0.679
4	87	222	151	1.558	0.742
5	336	257	1078	0.234	2.205
6	84	32	83	0.616	0.007
8	130	90	114	0.310	0.125
9	116	234	140	1.018	0.210
11	259	228	286	0.120	0.105
12	231	91	209	0.604	0.094
13	157	151	239	0.039	0.520
14	247	338	324	0.370	0.314
MARE				0.452	0.426
Variance				0.202	0.378

5.5.6 Effort Estimate using Unadjusted Function Points

Jeffery et al [53] and Kitchenham [63] report in their studies that the function point analysis technical complexity adjustment (TCA) did not improve the effort estimation model. The objective of this set of trials is to establish whether the estimation errors using unadjusted function points are different from the neural network models developed using function points as input. Again 12 networks were developed, and each of them were trained five times to select the models with the lowest MARE for each network. The results are given in Table 5.13.

A comparison of the MARE indicates that the neural network estimates using unadjusted function points as inputs resulted in a lower MARE and variance. The paired two sample for means t-Test rejects the hypothesis that the difference in prediction error is not equal to zero. This implies that the difference between the two approaches is not statistically significantly different in this dataset.

Table 5.13 Prediction Error Comparison using Unadjusted Function Points

Project No	Effort PM	NN Est (FP)	NN Est (unadj FP)	FP ARE	Unadj FP ARE
1	287	299	158	0.041	0.450
2	83	70	77	0.146	0.070
3	1107	697	361	0.371	0.674
4	87	222	115	1.558	0.328
5	336	257	245	0.234	0.272
6	84	32	74	0.616	0.117
8	130	90	247	0.310	0.898
9	116	234	170	1.018	0.465
11	259	228	350	0.120	0.353
12	231	91	228	0.604	0.010
13	157	151	212	0.039	0.353
14	247	338	247	0.370	0.000
MARE				0.452	0.333
Variance				0.202	0.073

5.5.7 Discussion

The objective of these trials was to assess the neural network effort estimation accuracy compared to the models assessed by Kemerer [60]. A restricted sample size places limitations on such an evaluation. With the approach taken within such a limited sample the neural network MARE and variance was not significantly different.

Hornik et al [46] had concluded that any lack of success in neural network applications must arise from either inadequate learning, insufficient hidden units, or a lack of a deterministic relationship between input and output. The cascade networks ensure that the number of hidden units are optimised. With the limited sample size learning may be inadequate, but apart from this the prediction error may just be a lack of a deterministic relationship between system size (function points) and development effort.

Jeffery et al [53] had concluded that system size attributes approximately 40 percent of the effort variation. In these trials the neural network MARE was approximately 40 to 50 percent, which suggests that a large proportion of the development effort is not

ascribable only to system size. The other development effort attributes or cost drivers may have to be modelled to produce more accurate estimates.

In these trials the MARE and variance of the networks using unadjusted function points were lower than those achieved by using function points which had been adjusted using the technical complexity adjustment factors. Even though on the sample used the difference was not statistically significant, further investigation is warranted to test whether neural networks are better able to model these technical adjustment factors than the function point analysis scheme.

5.6 Mermaid Data

5.6.1 Introduction

Prior research for this thesis (Chapter 2) showed that for commercial software development, productivity for those projects with the highest productivity may be at least ten times that of projects with the lowest productivity [123]. To enable accurate project effort estimation it is necessary to model those factors which influence productivity and thus development effort.

Jeffery et al [53] attribute approximately 40 percent of the effort variation to system size (function points). The neural network trials discussed in Section 5.2 above demonstrated a cascade network's ability to effectively model non-linear relationships from one dimensional space into its own weight space. The trials discussed in Section 5.3 demonstrated that neural networks can simultaneously model numerous development attributes to accurately estimate development effort in the relatively large simulated development dataset. In the models in Section 5.4 (Desharnis) and 5.5 (Kemerer) the neural networks were not able to predict development effort with the same accuracy as in the earlier sections, and the MARE was approximately 0.35. The models in Section 5.5 (Kemerer) used only system size as input, while those in Section 5.4 (Desharnis) in addition had only three other factors. These may possibly not have been the major factors affecting development productivity, and with many factors

excluded which may be significant in software development [55] the result is not unexpected. The limited sample size may in both cases also have contributed to the insufficient capture of the problem domain attributes by the neural networks.

In this section the MERMAID-2 dataset [62] is analysed using cascade networks. MERMAID is a joint collaborative project part-funded by the European Commission's ESPRIT program which aims to develop and automate improved methods of cost estimation. The MERMAID-2 project database comprises 30 projects from public domain data and data that cooperating companies have made available. This dataset differs from other project data in that information on 21 development attributes which are expected to influence productivity, as well as the individual function point adjustment factors are included for 28 of the 30 projects.

Th objective of the trials in this section is to assess:

- the accuracy with which neural networks can estimate development effort using function points as input
- the accuracy with which neural networks can estimate development effort using unadjusted function points as input, and to establish whether with neural networks the exclusion of the technical complexity adjustment has a significant effect on prediction accuracy
- the effect which the inclusion of some technical adjustment factors into the network model have on the prediction error
- the effect which the inclusion of additional development attribute factors have on the MARE of the neural network model

Initially four sets of trials were conducted to pursue these objectives. These were followed by three further trials to assess some aspects which resulted from the first four trials.

5.6.2 Mermaid-2 Research Data

The Mermaid-2 dataset comprises 30 projects [62]. The data is not totally complete, but for all projects the size of the system measured in function points is recorded, as

well as the total effort measured in development hours, and the project schedule measured in person-months. The minimum, maximum, mean and median values are given in Table 5.14.

Compared to the ASMA project dataset [123] the size of projects are relatively small, with an average size of 277 function points. With the function point and total effort median values being less than half their mean values it indicates a skewed distribution of smaller projects. Of significance is the large productivity variation, where for the most productive development the productivity was 44 times that of the project with the lowest productivity. This productivity range is not unusual in project data as was discussed in Chapter 2, but it will place an increased challenge on the neural networks to model the significant relationships.

Table 5.14 Mermaid-2 Project Data Summary

	Function Points	Unadjusted FP	Effort Hours	Schedule Months	Productivity FP/Hrs
Minimum	23	23	238	2.0	0.007
Maximum	1,507	1,408	48,230	35.0	0.310
Mean	277	271	7,443	10.9	0.064
Median	129	126	3,568	8.8	0.042

For all projects except 22 and 27 the individual 14 influential function point technical complexity factors (for details see Appendix A) were recorded. For the 21 development attribute factors the values were available for all projects except 16 and 22. These are measured on a five-point scale where 1 corresponds to a very strong negative influence, 3 corresponds to no special influence, and 5 to a very strong positive influence. Data collectors occasionally missed out a factor if they considered that it was irrelevant, when it should actually have been regarded as having a value of 3, and the project data has been adjusted accordingly [62]. A list of the 21 productivity adjustment factors or cost drivers are given in Appendix C.

5.6.3 Research Design

The dataset consisted of enhancement (maintenance) and new development projects. For two projects the project type was not recorded, and seven projects were new developments. To determine whether the enhancement and new development projects could be grouped into a single category based on their respective development productivity, a two-sample (assuming unequal variances) t-test was conducted, with a hypothesised mean difference of 0. As the t statistic was less than the critical t-value the hypothesis that the productivity is significantly different is rejected and it was decided to have just a single group of projects.

For each of the trials the data was divided into a training set and a test set. To ensure that the network models were not required to make estimates outside the range of values on which they had been trained, the three largest and three smallest projects in terms of largest function point size and development hours were excluded from the pool of candidates for the test set. There was some overlapping in the projects identified by these criteria, and three projects were common in both sets. This eliminated a total of 9 projects for possible selection as test sample. With a limited sample size and the need for a large training set, the decision was arbitrarily made to include only 5 projects in the test set. The test set of five was then randomly selected using the Microsoft Excel 5.0 random number generator.

As the test sample size is relatively small the possibility existed that any outcome may have been the result of chance circumstances applicable to the five selected test projects. To minimise this possibility three different sets of test samples were generated, and all network trials were thus replicated three times. In each case the network was trained, and the prediction and generalisation ability was then tested on the 'unseen' projects in the test set.

As in the previous trials the initialisation weights influenced the prediction error. This was evident especially with such a small sample size. As these weights were randomly generated each time, the prediction error varied slightly each time. As before each network was trained repeatedly and in each case only the best model was selected for comparison. This approach was necessary to minimise the result of chance initialisation weights resulting in an uncharacteristically large error for a particular network. For all

evaluations the MARE was used. The MRE is also generally given in the results to indicate whether the effort estimates were biased.

5.6.4 Research Project Analysis

To achieve the project objectives a series of seven trials were designed to conduct analyses on the research data.

5.6.4.1 Trial 1

In contrast to regression analysis where the model is developed from the project data for which certain probability confidence factors are generated, the confidence in neural network models is determined by their ability to generalise outside their training set. The initial sample size of 30 was reduced to 28 as for two projects the systems size in unadjusted function points was unavailable. This automatically limited the next trial with unadjusted function points to this sample size. Including the two projects in this trial would have precluded a comparison of the two sets of results.

Three network files were developed all with identical parameter settings. In all cases the only input was the project size and the output was development effort. Both inputs and outputs were scaled by calculating their natural logarithm and then applying a linear scaling in the range of 0.1 to 0.9. This scaling method gave good results in Section 5.3 (Simulated SPQR/20 data) and was selected as the preferred method for this section. Pilot trials conducted with different scaling on this data confirmed that this scaling resulted in the lowest MARE. The network parameter settings were also similar to those discussed and used in Sections 5.2 and 5.3 except where explicitly stated otherwise.

The results are given in Table 5.15. As in previous sections where the only input into the network was the function point size of a system, the MARE was relatively large. In view of Jeffery's [53] conclusion that other factors contribute significantly to the variability in development effort, their exclusion from any model increases the MARE. The differences in the MARE of the three test sets indicates that these network models do not generalise well across different data sets. This suggests that either the training

set is not large enough or the one input supplies too little information to facilitate sufficient training by the network to capture the problem domain adequately.

5.6.4.2 Trial 2

For this set of trials the same three training and test sets were used as for Trial 1. The scaling and all parameter settings were identical. The only difference between the two trials was that in this case the input was unadjusted function points. The results are given in Table 5.15.

A comparison of the results of the networks using adjusted function points to those using unadjusted function points indicates that unadjusted function point models result in a reduced MARE and variance, although this difference is not statistically significant. This may in part be due to the small sample size with large variability within each sample. This reduced, though not significant reduction in MARE and variance is a constant phenomenon in all trials where such comparisons were conducted (see Sections 5.5 and 5.7).

Table 5.15 Trial 1 and 2 Results

	Trial 1 MRE	Trial 2 MRE	Trial 1 MARE	Trial 2 MARE
Test Set 1	0.40	0.32	0.96	0.66
Test Set 2	1.06	0.55	1.51	0.89
Test Set 3	0.81	1.08	1.65	1.35

As there is no statistical difference between MARE of the network models which used function points and those which used unadjusted function points this result is agreement with that of Kitchenham [63] who found that the function point adjustment factors were inappropriate and had no reliable and consistent effect on the relationship between size and effort.

5.6.4.3 Trial 3

The objective of this set of trials was to assess whether the inclusion of the technical complexity adjustment factors (TCA) into the network model with unadjusted function points would reduce the prediction error. The above trials indicated that the inclusion of the TCA into the *function point model* (ie the use of adjusted function points) did not reduce the MARE of the neural networks.

The same three sets of training and testing data were used as for Trials 1 and 2. In all cases unadjusted function points were used as one of the inputs and development hours as the one output. Both of these were scaled by taking their natural logarithm and then scaling the result linearly in the range of 0.1 to 0.9. For all networks the same parameter settings were used as in the above trials.

The initial networks used in addition to the function point size the 14 technical adjustment factors (TCA) as input. These TCA are in the range of 0 to 5 (see Appendix A) depending to what degree the various factors exert an influence in any particular development. These factors were scaled linearly in the range of 0 to 1. No initial analysis of the effect of the various factors was done as the neural networks should identify which factors are relevant and generate the model accordingly.

The problem that arose from this approach was caused by too small a training set. With the inclusion now of 15 inputs, the networks curve-fitted the training data. This resulted in very low training errors but these networks did not generalise well. The problem of over-fitting was discussed in Chapter 3.

With just a limited dataset being available it was decided to identify only the factors which had a significant influence on development productivity and thus development effort. The importance of the individual TCA was determined by using analysis of variance. This was done by dividing the observations for each factor into two groups. In the one group were the projects in which the technical complexity factor was absent or only present with a very low degree of influence, as was indicated by the value of the factor being either 0 or 1. The other group comprised the rest of the observations and comprised all projects where the factor under investigation has a reasonable degree of influence. The analysis of variance was conducted by investigating the relationship

between the technology factor and the development productivity of the two groups in each case.

This analysis indicated that only three of the 14 TCA were significantly related to productivity. These were factors 1, 6, and 8, which were all significant at the level of $p < 0.01$. This results agrees with that of Kitchenham [63] who in a slightly different analysis also concluded that those three factors significantly influenced productivity. Neural network models were then developed having only these three TCA factors together with unadjusted function points as input. The results are shown in Table 5.16.

Table 5.16 Trial 3 Results

	MRE	MARE
Test Set 1	0.17	0.36
Test Set 2	-0.41	0.41
Test Set 3	0.38	0.42

With the exclusion of the other 11 factors the neural network estimate was not expected to be as accurate as may have been the case if it had been possible to model also the effect of the less significant factors. When the MARE of Trial 3 is compared to those of Trial 1 and 2 a reduction is noted.

Paired two sample for means t-tests indicate that the reduction in error is statistically significant. This indicates that the neural networks using unadjusted function points and only the three most significant factors to estimate development effort results in a lower MARE when compared to networks using function points which had been adjusted in the conventional manner using all 14 factors. Unfortunately the limited sample size precluded assessing the neural network's full potential at appropriately adjusting unadjusted function points using all 14 TCA to derive an effort estimate.

5.6.4.4 Trial 4

In this series of trials the objective was to assess the effect of the inclusion of the 21 productivity adjustment or cost driver factors (see Appendix C for details) on the

neural network estimation error. As in Trial 3 a similar problem presented itself in that the number of observations in the training set were too few to permit the inclusion of all these factors without the network curve-fitting the data. This would have degraded the network's generalisation ability and resulted in poor estimates in the test set. A similar approach was taken to that in Trial 3 in that the importance of the individual adjustment factors was assessed by investigating the relationship between each factor and the project development productivity.

In the Mermaid-2 dataset the productivity adjustment factor values range from 1 which corresponds to a very strong negative influence of that factor, to 3 which corresponds to no special influence, and to 5 which corresponds to a very strong positive influence. For the analysis of variance the observations were divided into two groups – those projects in which the factor exerted a negative influence and had values of less than 3, and those projects in which the factors had a positive influence and had values of greater than three. All projects which had a value of three, and in which the particular factor had no particular influence were excluded from this analysis. For each of the 21 factors a two sample t-test value was calculated to establish whether the productivity as influenced by the negative influence of the factor was significantly different from that of the positive influence of that factor.

Only two factors were identified to have had a significant effect on productivity. These were user involvement and the working environment. Three networks were developed with the same test sets as in the above trials. The network inputs were system size (function points), the three technical complexity adjustment factors which were identified as having a significant influence on development productivity, and which were used in Trial 3, and in addition the two development attribute (cost driver) factors which were also identified as having a significant effect on productivity.

It had been expected that with these two additional factors the model MAREs would reduce further, as the networks now had even more information on which to train. The networks trained very well, with low training errors. The problem that arose, even with only six inputs, was curve-fitting, resulting in poor generalisation on the test set. The limited sample size placed a severe restriction on network performance, as there were insufficient observations on which to train before the networks started over-fitting.

The process of developing the neural networks with the sample restriction became a juggling act to find the right balance. The trade-offs were giving the network additional information (inputs) to improve its learning capability, and poor generalisation because of too many inputs which result in curve-fitting.

This situation was not satisfactory as the software development environment is complex, with many factors influencing productivity. By omitting some of these factors from the model, prediction accuracy must of necessity deteriorate. It appears that for the dataset used for these trials the maximum number of inputs is approximately four, where the MARE appears to be minimised for the limited number of inputs (see Trial 7). Any further increase resulted in over-fitting.

5.6.4.5 Trial 5

To overcome the problem of over-fitting with the introduction of the development attributes, three network models, using the same test sets as in the above trials were developed. For this set of trials the only inputs were unadjusted function points, and the two significant development attributes of user involvement and working environment. The function point input and the output goal values were scaled as in Trial 4, and similar parameter settings were selected.

The results are shown in Table 5.17. Although the MARE is lower than in the models where the only input was function points, the error is higher than in those networks in Trial 3 with three technical complexity adjustment factors.

Table 5.17 Trial 5 Results

	MRE	MARE
Test Set 1	-0.42	0.67
Test Set 2	0.30	1.37
Test Set 3	0.24	0.84

5.6.4.6 Trial 6

As was noted in the introduction of this section, the projects in the dataset comprised both new developments and enhancements. The t-test had indicated that the difference in productivity was not significant, and thus the dataset was treated as one homogeneous group. In this set of trials the assumption of no significant difference between the new and enhanced projects was waived. Again three networks similar to those in Trial 2 were developed, with the exception of the inclusion now of an additional input to differentiate between new and enhancement projects. All other input and output scaling, and the parameter settings were similar to those in Trial 2.

The results of this trial are shown in Table 5.18, together with those of Trial 2 as a comparison. The MAREs in Trial 6 are lower than those in Trial 2, and the paired two sample for means t-test indicates that the reduction in error is significant. This result indicates that the criteria which has been used to determine which factors to include as network input may not be entirely appropriate, with the group variance being larger than between the groups. The initial t-test had indicated no significant productivity difference due to the type of project, but the neural network result using this input is significantly better.

Table 5.18 Results of Trial 6

	Trial 6 MRE	Trial 2 MRE	Trial 6 MARE	Trial 2 MARE
Test Set 1	0.03	0.32	0.26	0.66
Test Set 2	0.50	0.55	0.89	0.89
Test Set 3	0.50	1.08	0.68	1.35

5.6.4.7 Trial 7

With the inclusion of the project type having a significant effect on the MARE in Trial 6, another set of Trials was conducted to test the inclusion of this factor with some of the other networks. The lowest MARE was generated by the neural network models in

Trial 3. Three networks similar to those in Trial 3 were developed, with the only difference being the addition of the project type input.

The results are shown in Table 5.19. The overall prediction errors are not significantly different from those in Trial 3. With the addition of another input into the network model there was some suggestion during training of the neural networks that curve-fitting was becoming a problem. With a limited dataset this restriction prevented the network learning from the additional information provided by the added input. Without such restriction, additional relevant information should result in more accurate estimates.

Table 5.19 Results of Trial 7

	Trial 7 MRE	Trial 3 MRE	Trial 7 MARE	Trial 3 MARE
Test Set 1	-0.03	0.17	0.34	0.36
Test Set 2	-0.29	-0.41	0.29	0.41
Test Set 3	0.79	0.38	0.79	0.42

5.6.5 Conclusion

During various stages of some of the above trials the parameter settings, such as the weight range setting, candidate pool size, input and output Mu, and candidate patience were altered to assess whether this would result in a reduced prediction error. At no stage did these parameter setting alterations result in improved model performance. The settings as discussed in Chapter 3, and in this chapter, were thus maintained for all models which were used for these evaluations.

With the limited dataset in which development productivity varied by a factor of 44, the neural networks using system size measured in unadjusted function points, and three of the technical complexity adjustment factors, were able to predict with an average MARE of approximately 0.40. To develop network models which are able to

predict development effort more accurately it is essential to have a larger dataset on which to train the neural network, in addition to permitting the addition of other development attributes which may affect development effort. Without this the capture of the problem domain into the network weight space is limited. Adequate learning and capture of the problem domain is essential for good generalisation and estimation.

The trials indicated that the use of function points which had been adjusted by the TCA factors produced no lower MARE than unadjusted function points. This is in agreement with the results obtained by conventional statistical techniques [53] [63]. In general the restricted sample size limited the prediction potential of the neural networks. The small dataset is typical of empirical data and this may be a limiting factor in the development of neural network models by individual organisations on their own data. This also highlights the importance of cooperative datasets such as the ASMA project database [4].

Trial 6 indicated that network inputs can not be selected only on the basis of t-test significance. Here the networks had included an input (project type) which had been shown not to significantly affect productivity, but which resulted in a significant reduction in MARE. With a small dataset, the choice of the limited number of inputs becomes difficult, as factors may be interrelated. Finding the correct combination of inputs is difficult, being guided by tests of significance, but requiring a trial and error approach to further optimise the choice.

With the small dataset it was not possible to assess fully the effect of the cost drivers in a neural network model. There are many factors which affect development effort, and these vary from one development to the next. Inherent in development data are large differences in productivity. It is too ambitious to expect all this information to be hidden in so few observations. Neural networks are known for their ability to model complex and non-linear relationships found in data, but they are no magic wands which can produce something out of nothing. It is essential that the training sets contain sufficient relevant examples for the neural network to learn sufficiently from these to enable accurate estimates and generalisations to be made.

This set of trials showed the accuracy with which neural networks are able to predict effort from unadjusted and adjusted function points. They also showed how the model

was significantly improved by the inclusion of selected technical complexity adjustment factors. Within the limited number of observations the neural networks were not able to include cost drivers successfully to improve network performance.

5.7 ASMA Project Data

5.7.1 Introduction

The Australian Software Metrics Association (Victoria) has developed a database of mainly Australian software development project information. Terry Wright, the project database leader states that organisations in the United States and the United Kingdom have been working on a similar database for several years, but have made little progress. They have now decided to abandon their project databases and adopt the ASMA one exclusively [83].

In November 1994 the Release-5 [4] of this database was made available, and it is this set of project data which was used to assess the artificial neural networks' ability to accurately predict development effort

5.7.2 Project Data

The dataset contains information on 136 projects. Of these 80 were new developments, 47 system enhancements, and 9 were miscellaneous projects. These projects were developed on mainframe, mid-range and personal computers (PC). All except for one project which was developed in a 2GL (second generation language) were developed in either 3GLs or 4GLs. Twenty eight different development languages were used. The four most commonly used languages accounted for over 69 percent of the projects. These were PL/1, COBOL, Natural, and C with 30, 28, 27, and 9 percent respectively. Of the 136 projects 119 were DBMSs.

An attempt was made by the Australian Software Metrics Association (ASMA) to ensure the reliability and consistency of project data by developing a project data

collection package which was used for all project data. Currently the International Software Benchmarking Standards Group (ISBSG) version 1.0 collection package is being used. This is based on the ASMA model and has been adopted by the Metrics Associations of the USA, UK, New Zealand, Netherlands and Germany, and has now become the de facto international standard [4].

The development of the collection package is an attempt to ensure a consistent format to allow meaningful comparisons. Definitions are also provided for several measurements to reduce subjectivity and inconsistency in measurement. For the function point count the IFPUG standard [35] is adopted.

In addition to the function point size of each project, the time recording level as well as the time recording method are included. There are five time recording methods and the classification reflects the manner in which work effort is determined. There are four time recording levels. These reflect the scope of work effort which is included in the effort measure.

Level-1 comprises the work effort expended by the project team, project management, and project administration. Level-2 comprises in addition to Level-1, data base administration, data administration, quality assurance, data security, standards support, audit and control, and technical support. Level-3 in addition to Level-2 included software support, hardware support, and information centre support, while Level-4 in addition to all these also includes support to application users and clients, user liaison, and user training. In total 58.9, 31.6, 5.1 and 4.4 percent used time recording levels 1, 2, 3, and 4 respectively.

The variations in time recording levels complicate the analysis of this data. This makes comparisons of projects across time recording levels difficult. This time recording level division is of concern in an analysis as it further reduces the degrees of freedom.

An analysis of the project information indicated that for some projects the function point count may have been unreliable. For this reason 13 projects were removed from the data to be included in the analysis. Two outlier projects with extreme values in productivity were identified and excluded. Also excluded were all projects which were smaller than an arbitrary 30 function points, as it was considered that these were too small to cover the typical development scope. This eliminated another five projects.

One project (ASMA ID No 133) had required almost two times as much development effort as the next largest system. This was considered an outlier and excluded from further analysis. This meant that the number of projects was reduced by 21 to a total of 115.

The range of systems size, development effort and productivity are shown in Table 5.20. Across all three attributes the range is large. This complicates the development effort estimation process. For this dataset from which extreme outliers have been eliminated, the project with the highest productivity is approximately 34 times that of the lowest productivity. This range is typical of project development productivity (see Chapter 2, Section 2.4.2).

Table 5.20 ASMA Project Data Statistics

	Size Function Points	Effort Hours	Productivity FP/Hr
Min	31	65	0.033
Max	5,789	59,990	1.111
Mean	774	5704	0.252
Median	349	2388	0.175

In Table 5.20 the mean and median values for the three attributes are also shown. In all cases the mean exceeds the median value which indicates that the sample numbers are skewed towards the smaller or lower productivity projects.

Despite all the precautions taken to ensure reliable measurement of project data it is possible that some noise is present in the data due to the large scope of development environments and individual company practices and standards. Other sources of possible noise in the project data were discussed in Chapter 2, Section 2.3. Despite some possible difficulties with the project data it reflects typical data from which effort estimates have to be made in practice.

5.7.3 Research Methodology

With limited data the attempt was made to make the training set as large as possible. It was decided to use 10 projects for inclusion in the testing set. This left 105 projects in the training set. To avoid the neural network having to estimate outside the range of the data on which its was trained, the two smallest and largest projects in terms of function points and development hours were eliminated from the projects eligible for inclusion in the testing set. One project was common to both criteria and seven projects were thus not considered for inclusion into the testing set, and for all the trials remained in the training set.

To replicate the results three sets of training and testing data were selected, using the Microsoft Excel 5.0 random number generator to do this.

A literature search did not reveal any comparisons of the relatively recent cascade networks with the prediction capability of the popular back-propagation networks. The results of some comparison trials (Section 5.8.2) indicate that in many cases the cascade network performance as measured by the prediction error is approximately the same or better than that of back-propagation networks. For some reasons though which are not yet fully understood [25], the prediction error of cascade networks under certain conditions is significantly greater. For this reason in this set of trials all cascade network models were developed in parallel with back-propagation networks to monitor the respective prediction capabilities.

As in the previous research projects, the main measure of the software development effort prediction error in these trials was the MARE between the network estimate and the actual project effort. In the first set of trials the effort estimation error was determined using only the system size as input. In subsequent trials additional development attributes were added as inputs and the error measured to determine the effect these have on the development effort estimate.

5.7.4 Data Analysis

All analyses were conducted on the dataset of 115 projects as described above. To identify which project attributes have a significant influence on development productivity the t-Test for statistical significance was used.

5.7.4.1 Factors Affecting Development Productivity and Effort

There was no statistically significant difference in productivity between the new (64 projects), enhancement (44 projects), and miscellaneous (7 projects) projects. The projects were developed on three hardware platforms. These were mainframe platforms (82 projects), mid-range computers (19 projects), and PCs (14 projects). The t-Test for the mainframe hardware development platform indicated that productivity here was statistically significantly different from those developed on PCs. There was no statistically significant difference between productivity of projects developed on mid-range computers and PCs.

To test whether there was any difference in productivity between projects using different time recording levels two groups were formed. The one group included projects with time recording levels 1 and 2, while the other group included projects with time recording levels 3 and 4. The t-Test indicated that there is a statistically significant difference between the productivity of these two groups, ($\rho<0.01$).

Other factors such as the effect of the use of CASE, development methodology, DBMS application, use of application generators, and maximum development team size was not measured as for several of the 115 projects this data had not been recorded.

Using the Minitab Release 8 statistical software tool a regression analysis and stepwise regression were conducted to gain some insight into the linear relationships between some of the development attributes and effort. The independent variables were function points, development type, language type, time recording level, hardware platform, use of application generator, and DBMS, using the groupings discussed above. The dependent variable was development hours. For all the independent variables except function point system size binary encoded dummy variables were used. The results are shown in Tables 5.21 and 5.22.

Table 5.21 Multiple Linear Regression Results

Predictor	Coefficient	t-ratio	p
Constant	-1868	-0.65	0.519
Function Points	8.04	15.59	0.000
Development Type	254	0.22	0.823
Language Type	-1940	-1.69	0.094
Time Recording Level	863	0.48	0.635
Hardware Platform	2503	1.96	0.052
Application Generator	-481	-0.32	0.746
DBMS	-455	-0.28	0.780
R^2 = 74.3%			
R^2 adj = 72.6%			

Table 5.22 Stepwise Regression Results

Step	1	2	3
Constant	-418.5	-2811.7	-1466.3
Function Points	7.91	7.96	8.05
t-Ratio	16.71	17.40	17.69
Hardware Platform		3299	2579
t-Ratio		3.02	2.25
Language Type			-1947
t-Ratio			-1.86
R^2	71.2%	73.4%	74.2%

The coefficient of determination of the regression model was 0.74 and the only statistically significant attribute is systems size. With the F-statistic variable 'Entry' and 'Removal' level set at 3, the stepwise regression indicated that the development

hardware platform and language type are the next two attributes in terms of explanation of variance in development effort.

The development environment is more complex than the above analysis suggests. Not only are the relationships between project attributes and effort not always linear as was assumed above, but there may be interdependencies which were not modelled. For this reason, even though the above analyses suggested that apart from system size, the other project attributes may not significantly improve the explanation of the effort variation, they were all included for further analysis in the neural network models.

5.7.4.2 Network Parameters

The Cascade2 network parameters were set to the same value as those for the trials conducted in Section 5.3, and were only varied where this is stated. Numerous scaling techniques were assessed. Transforming the function point input and work effort development hours output by calculating their natural logarithmic values before scaling them in the range of 0.1 to 0.9 consistently gave the best results. This scaling technique was then used for the various analyses. The other input data was encoded either in a binary or scaled linear format.

For the back-propagation networks the same scaling technique was used as for the cascade networks. Preliminary trials on the project data indicated that setting the learning rate of the back-propagation networks to 0.3, and the momentum coefficient (Section 3.2.2) to 0.6 gave consistent good results. These preliminary trials also indicated that using only a single layer of hidden neurons tended to consistently give the lowest prediction error. Neuralyst permits the choice of the gaussian, sigmoid, linear and step-wise transfer functions. Even though the gaussian function performed well, the sigmoid function generally resulted in lower prediction errors on the project data and for this reason it was selected as the threshold function for all the trials in this section.

The network topology affects prediction accuracy as is discussed in Chapter 3 and Section 5.8.2.1. The topology has to be optimised for different training sets. The guidelines for doing this were discussed in Chapter 3 and these were followed for all trials using the back-propagation networks in this project.

To prevent over-training, the back-propagation network training sessions were interrupted every 100 epochs to predict project effort in the testing set. This estimate which was scaled, was normalised and the MARE for the testing set was calculated. Training was stopped when the MARE had reached it's minimum. To attempt to ensure that this minimum was not a local minimum to be followed by either another local minimum or the global minimum, the network training was continued until well past the stage where the prediction error had minimised.

5.7.5 **Trial 1**

In this set of trials the networks predicted development effort with only the single input of the function point size of projects. Many factors apart from the size of a system affect software development effort. Using only project size as input limits the information which the networks have to enable them to adjust their weight space, and effort estimation is not as accurate as when further relevant attributes are added.
The results of the cascade networks and the back-propagation networks for all three testing sets is given in Table 5.23. The MRE is also included to indicate to what extent the models on average over or under estimated development effort. For the cascade networks the MARE for the three test sets combined was 0.45, while for the back-propagation networks its was 0.29.

Table 5.23 Error Using Only Function Points as Input

	Test Set 1	Test Set 2	Test Set 3	Mean
Cascade				
MRE	-0.29	-0.03	-0.07	-0.04
MARE	0.42	0.48	0.44	0.45
Back-Propagation				
MRE	0.12	-0.14	-0.20	-0.08
MARE	0.44	0.22	0.21	0.29

The mean error for the three tests sets combined for the back-propagation models was lower than that of cascade network models. In particular the estimate in Test Sets 2 and 3 were considerably lower. In Test Set 1 the back-propagation model had a large ARE of 2.18 for one project. It is possible that this may be an outlier project. If this project is removed from the calculation of the MARE for Test Set 1, the error reduces from 0.44 to 0.25.

The back-propagation model prediction error for Trial 1 is significantly lower than that of the cascade model. For the back-propagation effort estimates, for all three tests sets combined, the percentage error smaller than 10 percent, 25 percent and 50 percent is shown in Table 5.24.

Table 5.24 Prediction Accuracy of Back-Propagation Networks

ARE Range	Percent	Cumulative Percent
0–10%	33.3	33.3
10–25%	23.3	56.7
25–50%	40.0	96.7
>50%	3.3	100.0

5.7.6 Trial 2

The initial statistical analysis had indicated some factors which may affect development effort. For this set of trials six factors are included one at a time with the function point measure as input into the neural networks, to establish whether neural networks can detect any relationship between this factor and function points, and development effort. The factors which were included in the models are shown in Table 5.25

Table 5.25 ASMA Project Attributes

Project Attribute	Categories
Language Type	3GL and 4GL
Time Recording Level	Levels 1–4
Hardware Platform	Mainframe, Mid-range, PC
Application Generator	Yes/No
DBMS	Yes/No
Development Type	New, Enhancement, Miscellaneous

Initial trials were conducted to establish the attribute encoding scheme to give the best results. Two approaches were evaluated. One was to use binary encoding similar to the use of dummy variables in regression analysis. This resulted in numerous inputs to accommodate this type of coding. The other approach was to use ordinal values in the range 0–1. For example for Time Recording Levels 1, 2, 3 and 4 the values used were 0.25, 0.50, 0.75 and 1.0.

There was no significant difference in the prediction error between the two coding schemes. It was decided to use the ordinal values scheme generally as this resulted in fewer inputs and thus fewer weighted connections. For a dataset with a limited number of observations this is less restrictive. The project attributes categorising whether an application generator had been used for development, and whether the project was a DBMS application, had several missing values which were not recorded. It was decided to include the attributes as inputs despite this, as it permitted the assessment of the robustness of neural network models in conditions where data is missing. For the DBMS attribute the coding used was 1 if it was a DBMS application, -1 if it was not, and 0 is it was unknown. A similar coding scheme was used for the application generator attribute.

The results of the cascade network models are given in Table 5.26, while those of the back-propagation models are given in Table 5.27. In each case the network had in

addition to the function point measure the attribute mentioned in the respective tables. For comparison the results using function points only are included in this table as well.

Table 5.26 MAREs of Cascade Models

Project Attribute	Test Set 1	Test Set 2	Test Set 3	Mean
Function Points	0.42	0.48	0.44	0.45
Language Type	0.46	0.51	0.30	0.43
Time Recording Level	0.47	0.50	0.55	0.50
Hardware Platform	0.30	0.83	0.35	0.50
Application Generator	0.40	0.44	0.36	0.40
DBMS	0.49	0.51	0.48	0.49
Development Type	0.37	0.84	0.45	0.55

Table 5.27 MAREs of Back-Propagation Models

Project Attributes	Test Set 1	Test Set 2	Test Set 3	Mean
Function Points	0.44	0.22	0.21	0.29
Language Type	0.35	0.27	0.21	0.28
Time Recording Level	0.38	0.29	0.39	0.35
Hardware Platform	0.28	0.26	0.23	0.26
Application Generator	0.45	0.22	0.21	0.29
DBMS	0.44	0.24	0.20	0.29
Development Type	0.30	0.22	0.22	0.25

The results show that the added attributes did not reduce the estimation error significantly and consistently for any attribute for both the cascade and the back-propagation networks, despite the additional information being provided. In some instances the error on some large outlier predictions had improved. For example in

Test Set 1, one project had a large ARE, and this was reduced considerably by the inclusion of the development hardware platform attribute, resulting in a reduced MARE for the set.

These results reflect some of the complexities of the software development environment, where some attributes have a different effect in the presence or absence of certain other attributes. This contributes to the complexity of the estimation model development. Because of this the result of no significant and consistent reduction in MARE is not entirely unexpected, and this emphasises the need for an integrated model to accommodate the necessary interrelationships.

5.7.7 Trial 3

These trials were conducted to test the overall models combining the seven development attributes mentioned above into an integrated model. The attributes in addition to system size (function points) are those shown in Table 5.25. By combining the attributes into a single model permits the neural networks to model the interrelationships which exist between the development attributes.

For both the cascade and back-propagation models the same parameter setting were used as in the above trials. The input coding scheme used was also the same as described in Section 5.7.6. Again the same three test sets were used. As in the above trials each cascade network model was trained on three different randomly generated initialisation weights and the model with the lowest prediction error was included in the result. The development of each back-propagation model required a systematic development approach as discussed in Chapter 3, to establish the topology resulting in the lowest prediction error. The back-propagation estimates for all three test sets combined are given in Appendix D Table D.4, and a summary of the results are given in Table 5.28. As in Trial 1 and 2 above, the cascade networks were not able to match the prediction accuracy of the back-propagation networks.

Table 5.28 Models Using Seven Inputs to Estimate Effort

	Test Set 1	Test Set 2	Test Set 3	Mean
Cascade				
MRE	-0.12	-0.06	-0.06	-0.08
MARE	0.41	0.28	0.41	0.37
Back-Propagation				
MRE	-0.11	-0.14	-0.09	-0.11
MARE	0.15	0.17	0.19	0.17

For the back-propagation effort estimates, for all three tests sets combined, the percentage error smaller than 10 percent, 25 percent and 50 percent is shown in Table 5.29. It is noted that in one project the prediction error was greater than 50 percent. For this project the error was 51 percent (see Appendix D, Table D.4).

Table 5.29 Prediction Accuracy of Back-Propagation Networks

ARE Range	Percent	Cumulative Percent
0–10%	40.0	40.0
10–25%	36.7	76.7
25–50%	20.0	96.7
>50%	3.3	100.0

5.7.8 K-Nearest Neighbour Estimates

The nearest neighbour concepts have made significant developments in the field of pattern recognition. While neural networks have been popular with researchers in recent years, the use of nearest neighbour concepts was restricted by excessive computational loads. Recent hardware advancements such as systolic arrays have been adapted for efficient nearest neighbour computations which have eased the

computational requirements considerably[19]. This has made the tool viable for practical applications.

The nearest neighbour concept was first proposed by Fix and Hodges in 1951 [19]. In 1970 Patrick and Fisher proposed a generalised family of k-NN rules based on the theory of statistical tolerance regions for two pattern classes [82]. There are some similarities between neural networks and nearest neighbour (NN) techniques and the type of application for which they are suited. For this reason a comparison was made to the neural network prediction to assess how this technique of pattern matching compares. The k-NN estimates were only conducted on Test Sets 1–3 of the models used in Section 5.7.7 above. The results are given in Table 5.30

Table 5.30 k-NN MARE Comparison

	Test Set 1	Test Set 2	Test Set 3	Mean
k-NN	0.32	0.28	0.23	0.28
Cascade	0.41	0.28	0.41	0.37
Back-Propagation	0.15	0.17	0.19	0.17

The k-NN uses the same header and data input files as Cascade2. The MARE for k-NN was 0.28 which lower than that for Cascade2, but does not match that of the back-propagation networks. The accuracy of k-NN estimate within 10, 25, and 50 percent is given in Table 5.31. The models which were developed did not meet the criteria set for an adequate estimation model in the research question.

Table 5.31 Prediction Accuracy of k-NN

ARE Range	Percent	Cumulative Percent
0–10%	23.3	23.3
10–25%	16.7	40.0
25–50%	43.3	83.3
>50%	16.7	100.0

As with neural networks the k-NN estimates are affected by the parameter settings. For the above models 12 nearest neighbours gave the best result. As neural networks were the main focus of this thesis the k-NN models were not tested to the same extent. The results above suggest that further research may be warranted to test a wider range of parameter settings and data scaling in an attempt to improve the estimation accuracy.

Another reason why the effort prediction capability of k-NN was assessed was because their output may be used as an input file into the neural networks. If the nearest neighbours generated by k-NN provide additional information it should enable the neural networks to improve their prediction accuracy. The main problem in assessing this capability on the project data was the limited number of observations in the dataset. When k-NN was set to only 2 nearest neighbours it generated 25 inputs for the neural network. By using 12 nearest neighbours, which gave the lowest prediction error, the number of inputs generated for the neural network would have been excessive for the 105 training set observations.

The 25 inputs generated by the 2 neighbour k-NN prediction did not improve the prediction accuracy of the respective neural networks. It was not possible because of the limited size of the dataset to adequately test this approach of using the output from k-NN for the neural networks, and no significant interpretation is possible.

5.7.9 Discussion

In Trial 1 the only input into the network models was system size. According to Jeffery et al [53] in their study approximately 40 percent of the variation in actual development effort is explained by function point size measure. This implies that many other factors which affect development effort were not included in the model. With only limited information on a dataset where productivity for the most productive project was 34 times that of the least productive, the back-propagation model MARE of 0.29 is considered reasonable. If the one outlier project in Test Set 1 which was discussed earlier were eliminated from the set, the overall MARE would be 22.6 percent, which is less than one of the criteria set for an adequate estimation model

[16]. On the other hand the percentage of predictions within 25 percent of the actual observations is only 56.7 percent.

The cascade networks did not perform as well as the back-propagation networks on this data. Attempts to improve prediction accuracy by adjusting the numerous parameter values discussed in Chapter 3 were not successful. Szabo [105] suggested reducing the weight range of the initialisation weights and increasing the size of the candidate pool to minimise the effect of the initialisation weights. Fahlman [25] considered that over-training may be a problem and suggested reducing the candidate patience and the output patience. None of these were successful in improving the cascade networks prediction performance to a level comparable to that of the back-propagation networks.

In Trial 2 the inclusion of individual attributes did not reliably improve the estimation error. Of interest is that not even the inclusion of attributes which were identified by statistical techniques to have had an effect on development productivity and development effort recorded a consistent reduction in error. In some cases a reduction in one test set is noted, but this reduction in error was generally not consistent across the other two test sets as well. Ideal training and testing sets should be larger than that which the ASMA project database Release 5 permit. Restricted by this limitation the sensitivity of results due to the inclusion of specific attributes under certain circumstances is noted. This is reflected by the inconsistent results across the different test sets.

When all seven development attributes are combined into a single model a reduction in MARE is recorded for both the cascade and back-propagation models. Combining the various attributes into a single model enabled the networks to model the interrelationships between the factors. As in all previous trials with this dataset the cascade effort estimate was not as accurate as that of the back-propagation models.

For the back-propagation models the MARE across all three test sets was 0.17 and in 76.7 percent of the cases the estimate was within 25 percent of the actual project effort, thus meeting both criteria set in the research question for an adequate model.

There are several other project attributes which are considered to have an effect on development effort [56] which were not included in the above models as this

information had not been recorded. It is desirable to record all relevant project data for inclusion in an estimation model, as without some of these factors the effort estimate will be less accurate.

5.8 Neural Network Issues

5.8.1 Network Stability

To assess the stability of Cascade2 in generating a consistent network topology to model the same underlying relationships, and to try and identify the source of any instability, two networks were developed on similar data. The dataset which was generated by SPQR/20 (Section 5.3) had a training set of 900 and a test set of 100. The data was simulated project development data.

A random number generator was used to divide the training set into two groups of 450 observations each. Even though for each of the two sets different randomly generated inputs were used to generate the simulated data, the underlying relationships between input and output were the same in both cases. The same test set of 100 was used for both networks, and the same randomly generated initialisation weight set was used.

The results of this trial are shown in Table 5.32. The results for both networks are similar. The only relatively large difference is in the number of epochs that were required to train the network. This difference may be the result of different initialisation weights of the candidate units each time the candidate pool is generated, resulting in differences in the time taken for network convergence to occur. A paired sample t-test conducted on the output or prediction values of both networks indicated that they were not statistically significantly different.

Table 5.32 Network Result Comparison

	Detail	Network 1	Network 2
Training	SSE	0.0106	0.0114
	RMSE	0.0048	0.0050
	Mean Error	0.0025	0.0028
Testing	SSE	0.0335	0.0345
	RMSE	0.0183	0.0186
	Mean Error	0.0115	0.0125
	Hidden Units	15	14
	Epochs	4171	3243

To assess the stability with which the networks captured the problem domain in their weight space, the weight values were examined. These values are given in Appendix D Table D.2, and are numbered 1–58 for Network 1 with 15 hidden units, and 1–57 for Network 2 with 14 hidden units. The weights may be divided into two categories. The first category are those weights from the input to the output units (1–43), and the second are the weights of the hidden units which were generated by the candidate pool (44–58).

As stated above, the initialisation weights of the input units were the same for both networks, but the Cascade2 network then generates different random weights for each candidate pool from which, after training, the best candidate is selected. The initialisation weights affect network performance, and therefore also that of the candidate units. Even though a candidate pool of 20 was used, the difference was still detectable, as can be seen by the differences in weights 44–58.

A visual inspection reveals some similarity between the two sets of weights, and this is particularly evident in the weights 1–43. The Pearson Correlation Coefficient which is higher for the weights 1–43 at 0.79 substantiates this. A paired two sample for means t-test for the complete set of weights indicates that there is no statistically significant difference between the two sets. This indicates some stability in the manner in which

the problem domain is captured from different data, but with the same underlying relationships.

The manner in which the problem domain is captured by artificial neural networks is reflected in the values allocated to the various weighted connections. To analyse to what extent the two networks allocate similar importance to the same input categories, and thus reflect stability in the interpretation of the problem domain, the Spearman Rank Correlation Coefficient (r_s) was calculated. The r_s for the full set of weights was not high at 0.52. The r_s for weights 1–43 (the input to output weights) was higher at 0.79, while the r_s for the weights of the hidden units was only 0.44. This low value for the hidden units indicates the instability introduced in the network architecture due to the differences in the initialisation weights of the candidate pool.

A visual inspection also indicates a similarity of weights for the different input classifications. Before any candidate hidden units were added the neural network had 43 weighted connections. Weight 1 (Appendix D Table D.2) is that of the network bias. The inputs into the network were eight classes, each with five categories, adding the next 40 weights. By dividing these 40 weights into sets of five, representing the five categories of each input of the eight classes, a pattern of similarity emerges within each group. This indicates that the network has identified differences in influence of the various inputs, and has assigned different weights which are similar for each input classification. As this trend is similar in both networks it suggests some stability in the manner in which the problem domain is modelled. The final two weights to bring the total to 43 are from two inputs which were generated by SPQR/20 and used in the model (Section 5.3.4).

This stability is not carried on to the hidden units, and results in different network topologies, requiring different numbers of training epochs. During the various network trials conducted for this thesis these differences were evident and resulted in differences in network performance. This instability in performance is of concern, as it is difficult to replicate studies with any large degree of accuracy. It is also of concern that an arbitrary number of network training runs have to be conducted for each trial, from which the best or average result may be selected. Further research to attempt to

eliminate this problem from cascade networks to achieve consistent optimum results is indicated.

Commenting on these issues Fahlman [25] states that because of anomalies such as these which are not yet fully understood the Cascade algorithm has not yet been published. Further development research to address these issues is being planned.

5.8.2 Back-Propagation Comparison

5.8.2.1 Introduction

Cascade networks have not yet been released for general distribution. They appear to have certain advantages, and especially their ability to optimise the network topology automatically was considered important for this research project.. Cascade networks have not been used as extensively as back-propagation networks and few comparative studies have been published.

A cross-section of comparisons was therefore conducted to provide some information on their relative predictive performances. To conduct this performance comparison the prediction error of the Cascade2 program used for this thesis were compared with results obtained using the Neuralyst software [80]. A difficulty with any such comparison is that network performance is affected by the parameter values which are used [74]. The types of parameter settings are not common to both tools in all cases, nor can it be assumed that the ideal setting for Cascade2 will necessarily be the ideal setting for Neuralyst, as the learning algorithms and the network architecture are somewhat different.

The effect of varying the network topology using Neuralyst back-propagation networks is illustrated by the trials conducted on the Desharnis dataset. The same data scaling, inputs, training and testing set were used as had been used for Test Set 1 in Trial 3 of Section 5.6.6. The results are shown in Table 5.33

Table 5.33 Network Topology and Prediction Error

Network Topology	MARE
4:12:1	0.69
4:16:1	0.36
4:20:1	0.53
4:12:4:1	0.43
4:8:8:4:1	0.36

In Table 5.33 the effect on the MARE due to different network topologies is large and the benefit of a dynamic self-organising architecture such as Cascade2 is evident. In any comparison it is difficult to be assured that the parameter settings have been totally optimised, as it is difficult to exhaustively test all combinations of settings.

5.8.2.2 Comparison Results

In addition to data used for this thesis the performance evaluation was conducted on one other dataset to obtain a wider comparison. Tan [106] had conducted a study using Neuralyst to develop an early warning predictor for credit union financial distress. The same data was used by this author (Wittig) to develop a network model using Cascade2. A comparison of the results is given in Table 5.34

Table 5.34 Financial Distress Data Comparison

Neuralyst	Non-Distress	Distress
Non-Distress	627	48
Distress	6	14
Overall Accuracy	92.23%	
Cascade2	**Non-Distress**	**Distress**
Non-Distress	653	22
Distress	6	14
Overall Accuracy	95.97%	

The results show that for Type 1 errors the prediction error was identical, being able to in both cases identify 14 of the 20 credit unions which subsequently suffered financial failure or distress. For Type 2 errors Cascade identified 653 against the 627 of Neuralyst. The overall accuracy of the cascade network was thus higher.

Five other comparisons were made using the results of analyses which were conducted for this thesis. The first of these assessments was made using the equation simulation of Section 5.2. The best result obtained for a Neuralyst developed back-propagation model was a MARE of 0.041, compared to the best CASCADE model of a MARE of 0.036. A similar result was obtained for the project simulation data from Section 5.3. For the Cascade2 model the MARE was 0.045 compared to the Neuralyst MARE of 0.048.

The next comparison used the Desharnis dataset from Trial 4 in Section 5.4. The best cascade network model had a MARE of 0.355, while the Neuralyst model had a MARE of 0.186. The comparison for the test set is shown in Table 5.35. The MARE of the Neuralyst model is significantly lower than that of the Cascade2 model.

Table 5.35 Comparison Results using Desharnis Dataset

Project No	Actual Effort	Cascade Estimate	Neuralyst Estimate	Cascade ARE	Neuralyst ARE
19	4494	4476	4442	0.00	0.01
26	3164	4260	2137	0.35	0.32
74	595	891	595	0.50	0.00
70	1155	1236	1523	0.07	0.32
37	1435	569	1174	0.60	0.18
79	9520	7769	11814	0.18	0.24
30	3948	5532	3877	0.40	0.02
61	2926	3763	3287	0.29	0.12
52	3136	2239	3577	0.29	0.14
47	4004	7517	5939	0.88	0.48
MARE				0.36	0.19

Another comparison was conducted using the Mermaid dataset (Section 5.6), Trial 3, Test Set 1. The Cascade2 model resulted in a MARE of 0.36, while the Neuralyst MARE was 0.35. The final comparisons were done with the ASMA project data. In most trials on this project dataset the cascade network models were not able to achieve the same prediction accuracy of the back-propagation networks. In the models in Trial 3 in which the networks had seven inputs of project attributes the MARE for the three test sets was 0.17 for the back-propagation models compared to 0.37 for the cascade network models.

Attempts were made to improve prediction accuracy of the cascade networks. Szabo [105] suggested reducing the weight range of the initialisation weights and increasing the size of the candidate pool to minimise the effect of the initialisation weights. Fahlman [25] considered that over-training may be a problem and suggested reducing the candidate patience and the output patience. None of these were successful in improving the cascade network's prediction performance to a level comparable to that of the back-propagation networks.

Several adjustments of the Cascade2 parameter settings were made. The effect of using the various activation functions was assessed, as was changing the values of the output and candidate epsilon, decay, Mu, prime offset, and change threshold values. None of these resulted in a reduction of the prediction error comparable to the back-propagation models.

5.8.2.3 Discussion

In several comparisons the cascade network prediction error was approximately the same or slightly lower than that of the back-propagation neural networks. The reason though for the Cascade2 algorithm not yet having been published is because of anomalies as were evident in the models developed on the Desharnis and ASMA datasets. The reasons why in some datasets the cascade algorithm does not perform as well as the back-propagation are not yet fully understood and may be linked to over-training, but this will require further research effort to eliminate the problem [25].

5.9 Conclusion

Six sets of trials research projects were designed to assess a range of neural network estimation problems. In the development of artificial neural network estimation models, the requirement is for them to capture the relevant relationships between project attributes and development effort from the project data into their weight space during the training sessions. In the first project the neural network's ability to extract a measurable function from one finite dimension to another was demonstrated. In the second research project the neural network estimation capability was assessed on a relatively large dataset of simulated project data. The networks demonstrated their ability to accurately forecast development effort in this environment, which is likely to have contained less noise than typical actual project data.

The availability of reliable large historical project databases is restricted. The next three research projects were designed to assess the estimation capability on three research project datasets. These datasets contained fewer observations than are required for neural networks to ensure adequate generalisation. In the Kemerer and Desharnis datasets many of the attributes which may affect development effort [56] were not available. As they represent datasets though which are typically used for metrics research it was decided to assess the neural network estimation performance under these restricted conditions. In all trials the estimation error was unsatisfactory and exceeded the criteria set for adequate model performance. The restrictions which the datasets imposed on neural networks did not permit the development of models which were capable of consistently and accurately estimating project effort.

The final research project utilised the Australian Software Metrics (ASMA) project database to develop estimation models. Several models with various project attributes as inputs were developed. Even though the dataset did not fully meet the neural network requirement, the best models were able to estimate development effort within 25 percent of the actual effort in more than 75 percent of the estimates, with an overall MARE of less than 25 percent.

The neural network performance during the research projects identified some areas of concern. One such concern was the lack of a consistent similar learning pattern in the weight space in similar project data with the cascade networks. Attempts were made to

reduce the effect of the randomly generated initialisation weights. Even when two neural networks start their training sessions on similar initialisation weights, there is no control over the initialisation weights generated for the candidate pool. The result is that under such conditions the optimum network topology which cascade networks develop are somewhat dissimilar, resulting in differing prediction accuracy, despite being trained on the same data, using the same starting initialisation weights.

In comparison trials between the back-propagation and cascade models the results were inconsistent. Despite the cascade networks on several datasets being able to generate estimates with a prediction error which was comparable to or slightly smaller than those of the back-propagation models, on some datasets the error was significantly larger. Identifying the source of this was outside the scope of this thesis. Further research is necessary to enable this to be corrected or that the reasons for this be identified to enable such conditions to be avoided.

Chapter 6

Software Program

6.1 Introduction

Most of the neural network research for this thesis was conducted using Cascade2 version 1.03 software [25]. To run this, a network file has to be developed first to set the various parameter values, as well as input the data file in a special format. For the thesis this was done by formatting the data appropriately in Microsoft Excel 5.0, before exporting it to Microsoft Word 6.0, for further changes and attachment to the file header section. The file was then saved as the network file in an ASCII text format.

The interface of the Cascade2 program is not very user-friendly, particularly when compared to Neuralyst [80], a artificial neural network program which runs under the Microsoft Windows environment. All the above factors would restrict the general use of Cascade2 as an estimation and management tool. A program was developed to provide a user-friendly interface, automate many of the functions, and set some of the parameter values to default values to reduce the overall complexity of the tool. This simplification has come at the cost of some program flexibility, but for its intended use it provides sufficient options, with the benefit of use of ease.

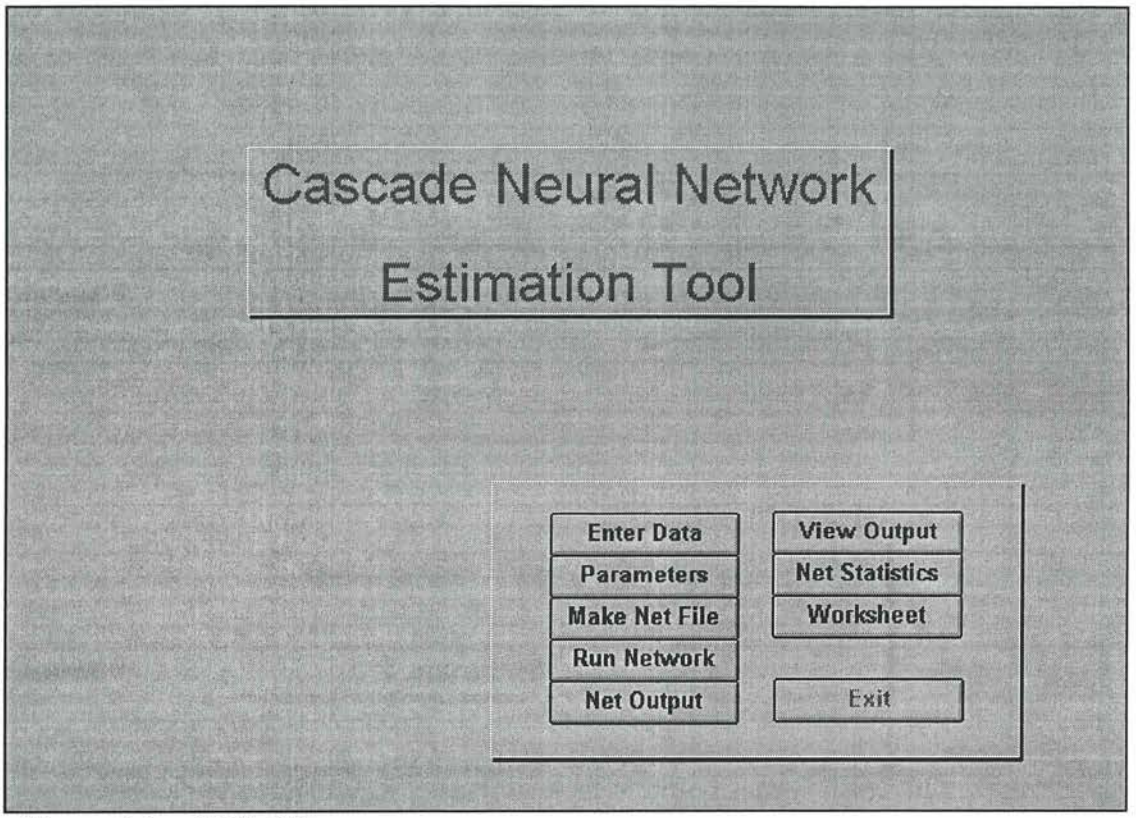


Figure 6.1 Main Menu

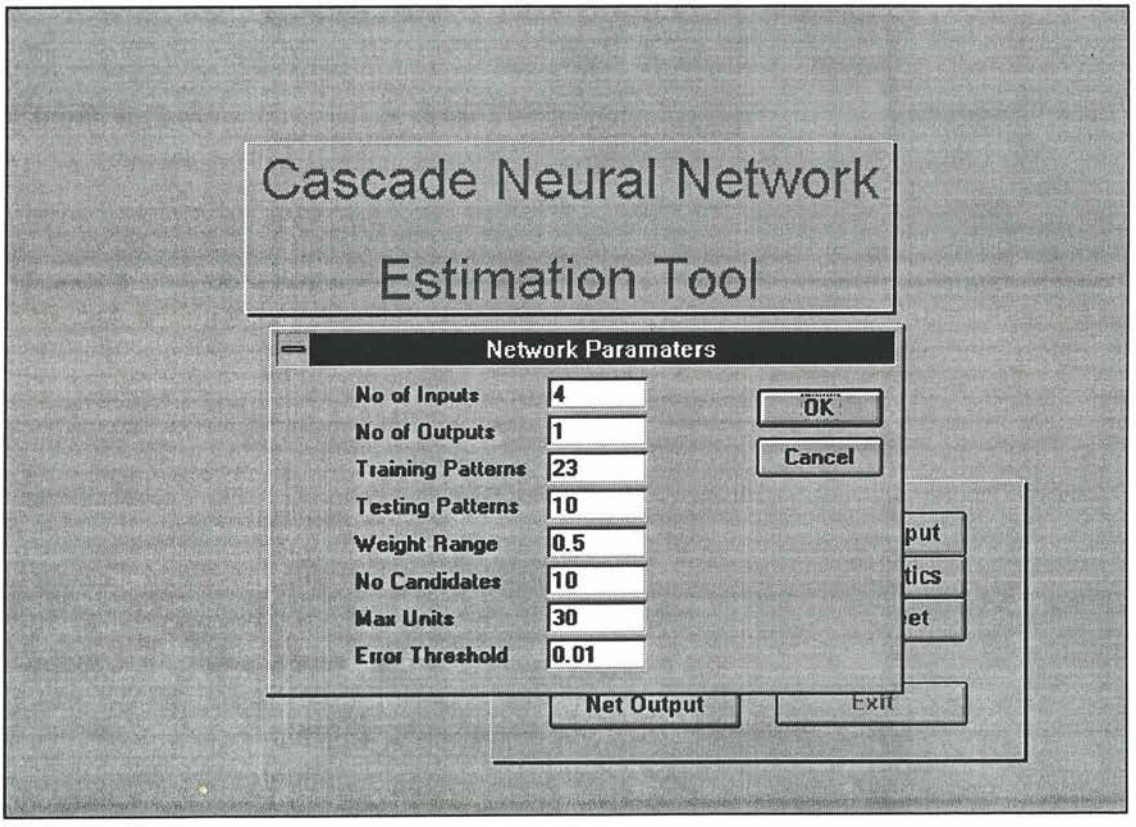
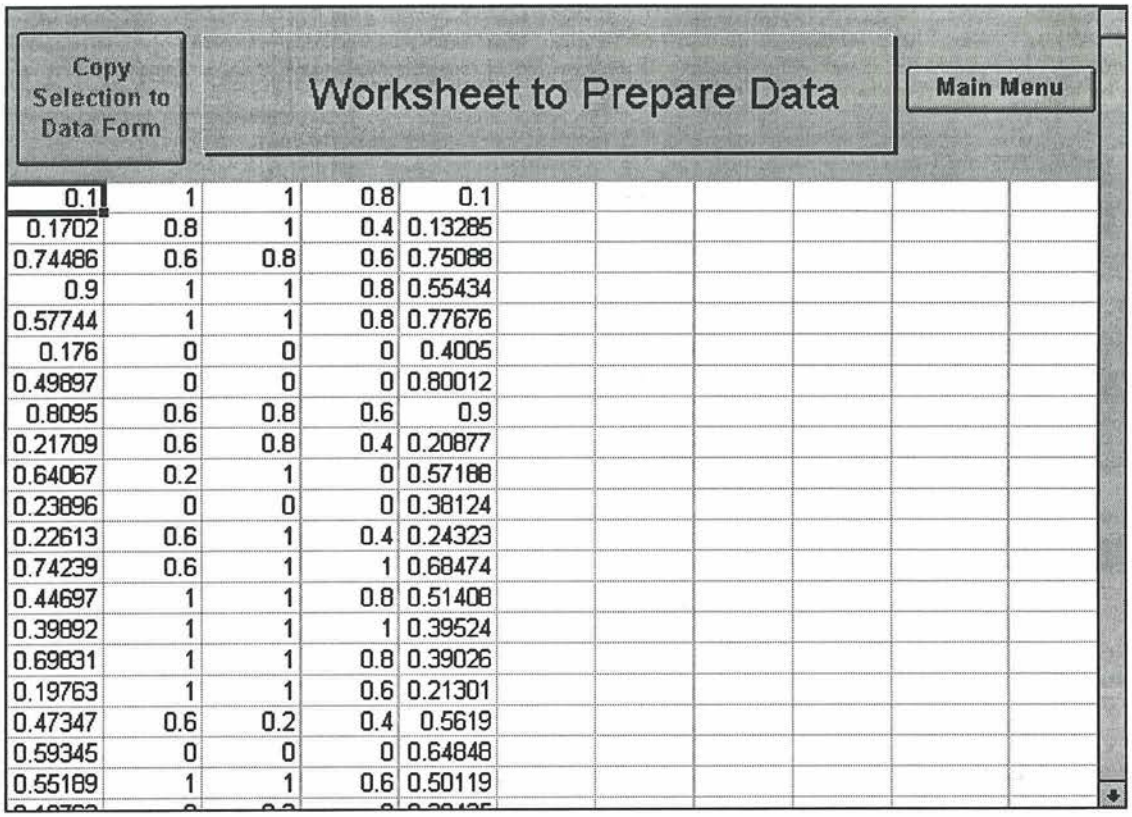
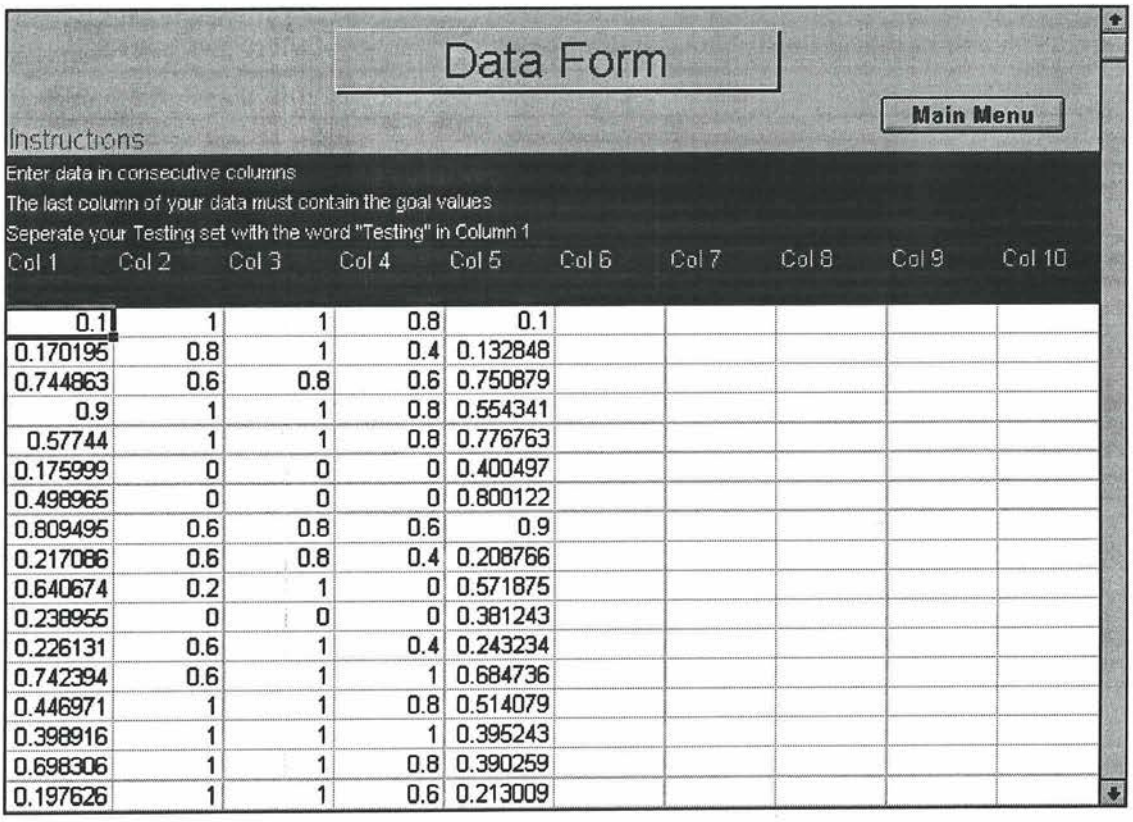


Figure 6.2 Parameter Setting Dialogue Screen



0.1	1	1	0.8	0.1
0.1702	0.8	1	0.4	0.13285
0.74486	0.6	0.8	0.6	0.75088
0.9	1	1	0.8	0.55434
0.57744	1	1	0.8	0.77676
0.176	0	0	0	0.4005
0.49897	0	0	0	0.80012
0.8095	0.6	0.8	0.6	0.9
0.21709	0.6	0.8	0.4	0.20877
0.64067	0.2	1	0	0.57188
0.23896	0	0	0	0.38124
0.22613	0.6	1	0.4	0.24323
0.74239	0.6	1	1	0.68474
0.44697	1	1	0.8	0.51408
0.39892	1	1	1	0.39524
0.69831	1	1	0.8	0.39026
0.19763	1	1	0.6	0.21301
0.47347	0.6	0.2	0.4	0.5619
0.59345	0	0	0	0.64848
0.55189	1	1	0.6	0.50119

Figure 6.3 Data Preparation Worksheet



Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8	Col 9	Col 10
0.1	1	1	0.8	0.1					
0.170195	0.8	1	0.4	0.132848					
0.744863	0.6	0.8	0.6	0.750879					
0.9	1	1	0.8	0.554341					
0.57744	1	1	0.8	0.776763					
0.175999	0	0	0	0.400497					
0.498965	0	0	0	0.800122					
0.809495	0.6	0.8	0.6	0.9					
0.217086	0.6	0.8	0.4	0.208766					
0.640674	0.2	1	0	0.571875					
0.238955	0	0	0	0.381243					
0.226131	0.6	1	0.4	0.243234					
0.742394	0.6	1	1	0.684736					
0.446971	1	1	0.8	0.514079					
0.398916	1	1	1	0.395243					
0.698306	1	1	0.8	0.390259					
0.197626	1	1	0.6	0.213009					

Figure 6.4 Data Entry Form

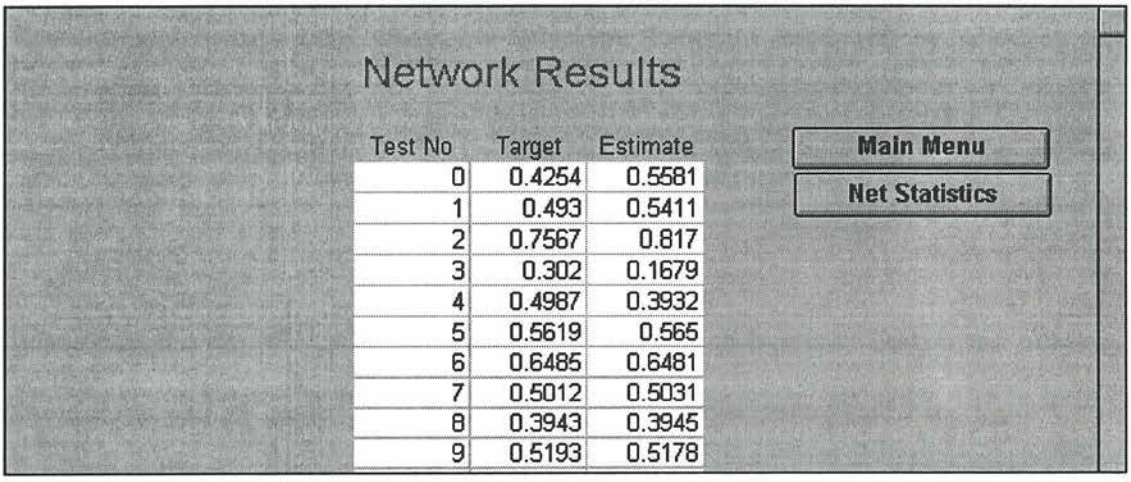


Figure 6.5 Network Output

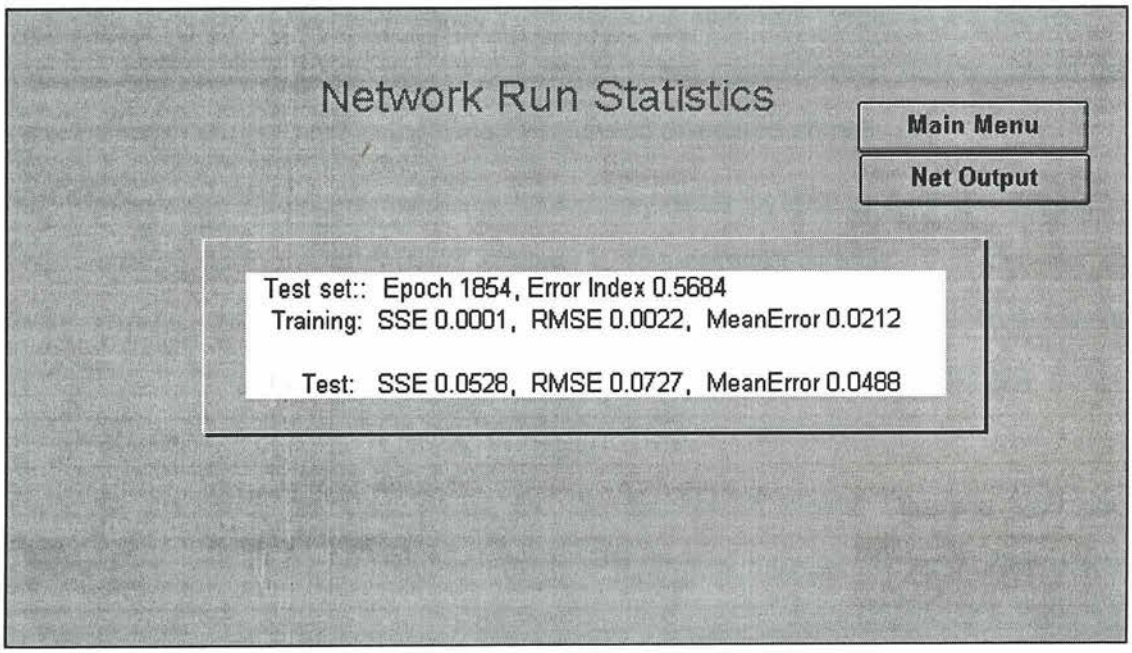


Figure 6.6 Network Statistics

6.2 Program Function

The program is activated from the Windows Program Manager by selecting the appropriate program icon. This automatically opens Microsoft Excel 5.0, removes the standard and formatting tool bars, status bar, formula bar, workbook tabs, and the vertical and horizontal scroll bars, to provide the opening main menu interface as shown in Figure 6.1.

Control of the program is through a series of command buttons. To run the program the first action would be to enter the input data for the neural network. If this data is available it may be copied or entered directly into the Data Form (Figure 6.4), which is activated by selecting the appropriate button on the main menu interface. Where necessary the vertical scroll bars are returned to screens, to permit scrolling beyond the screen limits.

Alternatively the data may be prepared in a special worksheet (Figure 6.3), again activated from the main menu panel. After the data has been prepared for input, it is selected and the 'Copy Selection to Data Form' button is activated. This results in the selected data being copied to the Data Entry Form discussed above.

The inputs in the data form may be entered in any order as long as they are entered in consecutive columns, starting from column 1. The network is able to predict one or more outputs, and the goal values are entered in consecutive columns after the input data. The training data set is separated from the testing set by inserting a line separating these two, with the word 'Testing' written in column 1.

Once the data is in the Data Form, and the number of Inputs, Outputs, Test and Training Patterns are known, this information is entered into the Network Parameters Dialogue box, which again is activated from the main menu. An example of a Network Parameters Dialogue box is shown in Figure 6.2. Only a few important settings are included here. The others have been preset to default values either through the file header which this program generates automatically, or through a special input text file, which is shown in Appendix E.

The values in the dialogue box are controlled by an 'Event Handler'. The initial values are those from the current network file header. The number of inputs, outputs, training and test patterns have to reflect those of the data in the Data Form (Figure 6.4).

The next step is to generate the network input file, by combining the file header with the data in an ASCII text file. An example of a simple network file with four inputs and one goal value, with 23 training patterns and 10 testing patterns is given in Figure 6.7. By selecting the 'Make Net File' control button from the main menu panel, the program automatically generates such a similar file. This is achieved by passing the new file header with the updated parameter values discussed above, and combining this with the data in the Windows Notepad editor, and then saving it as the INPUT.NET text file.

```
#INPUT.NET
NInputs 4
NOutputs 1
NTrainingPatterns 23
NTestPatterns 10
UnitType SIGMOID
OutputType LINEAR
ErrorMeasure INDEX
Graphics TRUE
WeightRange 0.5
UseCache TRUE
Test TRUE
MaxUnits 30
NCandidates 10
ErrorIndexThreshold 0.001
CandidateChangeThreshold 0.01
OutputChangeThreshold 0.01
COutputEpsilon 10.0
CInputEpsilon 100.0
CInputMu 2.0
COutputMu 2.0
OutputEpsilon 0.1
OutputMu 2.0
CandidatePatience 20
Outputpatience 20
```

```
Training
0.1 1 1 0.8 0.1
0.170195 0.8 1 0.4 0.132848
0.744863 0.6 0.8 0.6 0.750879
0.9 1 1 0.8 0.554341
0.57744 1 1 0.8 0.776763
0.175999 0 0 0 0.400497
0.498965 0 0 0 0.800122
0.809495 0.6 0.8 0.6 0.9
0.217086 0.6 0.8 0.4 0.208766
0.640674 0.2 1 0 0.571875
0.238955 0 0 0 0.381243
0.226131 0.6 1 0.4 0.243234
0.742394 0.6 1 1 0.684736
0.446971 1 1 0.8 0.514079
0.398916 1 1 1 0.395243
0.698306 1 1 0.8 0.390259
0.197626 1 1 0.6 0.213009
0.473466 0.6 0.2 0.4 0.561896
0.593449 0 0 0 0.648483
0.551891 1 1 0.6 0.501194
0.197626 0 0.2 0 0.394349
0.217086 0.6 0.2 0.2 0.519253
0.398916 1 1 0.8 0.596739
Testing
0.402516 0.8 0.4 0.4 0.425427
0.412935 0.6 0.2 0.4 0.492992
0.70929 0.6 0 0 0.756651
0.254837 1 1 0.6 0.302004
0.717227 1 1 0.8 0.498749
0.473466 0.6 0.2 0.4 0.561896
0.593449 0 0 0 0.648483
0.551891 1 1 0.6 0.501194
0.197626 0 0.2 0 0.394349
0.217086 0.6 0.2 0.2 0.519253
```

Figure 6.7 Network Input File

The network is now ready to be run and is activated by selecting the appropriate button from the main menu. Cascade2 is a non-Windows DOS program and is run by means of a simple batch file. The network requires further inputs, which are automatically given by a text file (INPUT.TEXT) with preset values. These values were selected from experience gained by the author from running the Cascade2 networks, and should be acceptable for most applications.

Depending on the hardware used, as well as the size of the network file, the data characteristics and selected parameters, the network may train from a few minutes to several hours or days. During this time the cascade network automatically builds the appropriate network topology, and stops when the error does not decrease further, or has reached the set error threshold limit. With values provided by the INPUT.TXT file the network then generates an output file called TEST.DMP. A small example of such an output file is given in Figure 6.8.

This format is not suitable for easy analysis as for example on a spreadsheet. By selecting the 'Net Output' button from the main menu, the Windows Notepad is activated and automatically loads the TEXT.DMP file and strips out unnecessary text, and spaces, and also replacing some appropriate spaces with tab markers. This is then copied into the main program, which presents the network output as is shown in Figure 6.5.

This output is in a form where it permits either visual examination or it may be copied into a spreadsheet for further analysis if this is required. The network run statistics (Figure 6.6) may be viewed by selecting the appropriate button either from the network output display, or from the main menu.

TEST.DMP

Test Pattern 0
Goal: 0.4254
Output: 0.5056

Test Pattern 1
Goal: 0.4930
Output: 0.5648

Test Pattern 2
Goal: 0.7567
Output: 0.9360

Test Pattern 3
Goal: 0.3020
Output: 0.1560

```
Test Pattern 4
  Goal: 0.4987
Output: 0.3883

Test Pattern 5
  Goal: 0.5619
Output: 0.5730

Test Pattern 6
  Goal: 0.6485
Output: 0.6383

Test Pattern 7
  Goal: 0.5012
Output: 0.5084

Test Pattern 8
  Goal: 0.3943
Output: 0.4081

Test Pattern 9
  Goal: 0.5193
Output: 0.5015

************************************
 Test set:: Epoch 3933, Error Index 0.7061
  Training:  SSE 0.0034,  RMSE 0.0122,  MeanError 0.0282

     Test:  SSE 0.0815,  RMSE 0.0903,  MeanError 0.0648
************************************
```

Figure 6.8 Network Output File

6.3 Conclusion

This program integrates various tools required to use the Cascade2 neural network, by means of a user-friendly Windows-based interface. Some tasks have been automated and in other cases control has been simplified.

Some program flexibility is sacrificed at the expense of reduced complexity, by letting the program set default values where it was considered appropriate to do so. This reduced flexibility may not suit some network operators, but should be sufficient for it to be used as a business tool by project management for software development effort estimation purposes.

Chapter 7

Discussion and Conclusion

7.1 Introduction

The overall thesis consisted of six research projects. Each of these assessed specific aspects which were aimed towards achieving the overall objective of determining whether artificial neural networks are capable of providing adequate development effort estimation models.

In this chapter the various results are considered and an assessment is made of the contribution of these to the research objective.

7.2 Discussion of Research Findings

The significance of the results of each of the six research projects is discussed first, followed by an overview and a report on some specific issues.

7.2.1 Modelling a Known Function

In the first research project the objective was to test whether the neural network model with the parameter settings and scaling techniques which were used was able to extract underlying interrelated non-linear relationships from the input data.

The underlying relationships of the input data were not trivial, and with only two inputs the neural network had to capture these in its weight space. The range across which it had to predict was large in that the target productivity ranged from 0.0002 to 1.3668 function points per development hour. For the test data six system sizes, across an average team size ranging in each case from one to fifteen, were selected. The system sizes in the test set were thus the same as in Figures 2.7 and 2.8 which were used in the original research [32]. Even though the test data was not selected at random, there is no reason to believe that the neural network would not have been able to predict with similar accuracy on a randomly selected test set.

The neural network MARE was 0.036 and a t-Test indicated that the estimated productivity was not statistically different from the actual productivity as determined by the equation. It is therefore concluded that the neural network succeeded in extracting the equation function underlying the input data into its weight space.

Even though Hornik et al [46] claim that neural networks are capable of approximating any measurable function from one finite dimensional space into another, it was considered important to demonstrate this fact. Neural networks unlike some statistical techniques, do not provide probability or confidence coefficients with which to interpret results. In neural networks prediction confidence is reflected by the error in the test and validation sets.

Hornik et al [46] claim that neural networks can model *any* measurable function. This research project established that the neural networks were able to model a specific non-linear function well and did so by modelling the underlying relationships correctly. An inability to demonstrate this would have seriously questioned the neural network architecture, topology, learning algorithm and parameter settings which were used for this thesis.

7.2.2 Simulated Project Data

Neural networks model a problem domain in their weight space. The complex software development environment requires numerous weighted connections to capture the problem domain as was demonstrated repeatedly in several trials discussed in Chapter 5. Baum and Haussler [10] suggest that neural networks require a certain minimum

number of observations, based partially on the number of weighted connections. This implies that complex domains such as project development require many observations for satisfactory generalisation. This author is not aware of any project database which meets the requirements of adequate reliable data with sufficient observations to fully satisfy the neural network requirements.

To overcome this problem two approaches, each with specific trade-offs were taken. On the one hand a large dataset was developed using simulated project data. This has the advantage of having a complex environment with many project attributes to be modelled and with sufficient observations. It has the disadvantage that it is not possible to prove that all the relationships which are inherent in all project developments have actually been included in the simulated data. This implies that the network results may only be extrapolated onto projects similar to those of the simulated dataset.

On the other hand the capability of the neural networks was assessed on project datasets with limited numbers of observations. This had the advantage of assessing the neural network capability on actual project data, but due to the limited number of observations and recorded project attributes which limit network learning, the full potential of the neural networks could not be established.

To generate the simulated project dataset the project estimation tool SPQR/20 was used. In the Jeffery Low and Barnes [53] study SPQR/20 estimated project effort with a MARE of approximately 12 percent. This is interpreted as the tool having the capability to model many of the relationships which are inherent in software project development to enable it to make effort estimates with such reasonable accuracy.

SPQR/20 was therefore used to generate a dataset of 1000 observations to simulate project data. Neural networks are trained on project data and presuming there are sufficient observations, their learning ability is limited by the number and relevance of inputs representing project attributes which they are afforded. Project development is complex and many factors influence the effort required to develop a system. Unless a neural network also has access to these factors through its inputs it will not be able learn and capture the effect these have.

The inclusion of several project attributes complicates the modelling process, as not only are the relationships sometimes non-linear, but the attributes may not be

independent and may be interrelated. The availability of a large dataset of simulated project data which contains several development attributes enables the assessment of the neural networks' ability to model relatively complex development environments.

As in the previous case above, the estimation process is complicated by the large range in system size, total effort, and development productivity. Despite this the neural network was able to generate an estimate which was not statistically significantly different from the 'actual' project values. This indicates that given a sufficient number of observations, the neural networks were able to model the complexity of several project attributes to accurately estimate development effort. The neural networks had no problems in converging in the trials and this suggests that they may be capable of modelling considerably more complex project data with many more attributes, given a sufficient number of observations.

7.2.3 Results with Desharnis Project Data

One objective of the research project with Desharnis' data was to assess the neural network estimation error from data with limited observations and development attributes. According to Jeffery [53] system size may attribute as little as 40 percent of development effort variation. With limited attributes available as inputs to the neural networks it is not surprising for them to attain an MARE of 0.36.

The effect of providing the network with more relevant information is illustrated by comparing the results of Trial 1 with those of Trial 4 using the Desharnis project data, where the additional input of average development team size reduced the MARE from approximately 0.52 to 0.36. It is possible that with the input of further factors which affect development effort that the MARE may be further reduced.

Artificial neural networks are sometimes considered as a black box, in which certain inputs produce specific outputs. The perception is partly due to the lack of complete understanding of the combined interaction of the network weights and transfer function. Despite this suggestion of a black box, neural networks are not capable of black magic, and the networks can not produce good results from insufficient inputs. The results of a neural network can thus be no better than the quality of the inputs.

In this research project the effect of varying some parameter values was also examined. One set of trials increased the candidate pool to 100. This should have ensured a reasonable probability that a good candidate was available for selection each time, and this should have reduced the effect of the initialisation weights. Despite this the networks with a candidate pool of 100 performed no better than the default setting used for this thesis of 20, and still resulted in some variability of results with each training session. Further research may be necessary to improve the network learning algorithm to produce consistently good results.

7.2.4 Model Comparison

The research project using the Kemerer dataset was severely restricted by the limited sample size. No generic neural network estimation model was available either commercially or from previous research and this meant that a neural network first had to be developed. In this case the training set used could have influenced the performance of the neural network in the comparison. The dual approach taken in Section 5.5 of first using the rolling rotational training set method and finally comparing the results with two neural networks which were developed using other project training sets was therefore considered justified.

The output of the two neural networks trained on the Desharnis and Mermaid datasets reveal a consistent MARE. The comparison results would thus have been the same, immaterial which approach was taken. The rolling rotational approach was preferred as this result enabled a direct comparison to the two subsequent trials using lines of code in the one instance and unadjusted function points in the other.

The results of the neural network comparison to the model performance in Kemerer's study [60] were not conclusive. Despite consistently lower MAREs, the large variance within the small samples resulted in the t-Tests showing that this difference was not statistically significant. A larger dataset is a prerequisite to a meaningful comparison.

7.2.5 Estimation Using Unadjusted Function Points

Trials were also conducted to compare the effort estimation error using either only function points or unadjusted function points as input. In the Kemerer dataset even though the MARE of the network using unadjusted function points was 0.33, against 0.45 for adjusted function points, the improvement was not statistically significant. A similar result was obtained with the Mermaid dataset, where the MARE for all three trials was 0.97 for unadjusted function points compared to 1.37 for adjusted function points. In the trial these results show consistently reduced MAREs using unadjusted function points, but within the limited sample size this difference was not statistically significant.

This result is significant in that the technical complexity adjustment factor is a size driver and should adapt the size measure to reflect differences in functionality to the user as a result of technical complexity. With this adjustment the size measure is refined to reflect its 'true' value taking into consideration the complexity factors. Intuitively it would be expected that using the 'true' size measure should enable a more accurate estimate of development effort to be made. This was not the case when neural networks were trained on the datasets used in this project.

In the Mermaid dataset the comparison the MARE of networks using only the function point system size as input compared to networks using only unadjusted function points was lower but the difference was not statistically significant. When three of the technical complexity adjustment factors were added as input into the neural network together with unadjusted function points, the reduction in error was statistically significant. In this case aggregating all three test sets the MARE had reduced from 0.97 to 0.40.

Relevant here is that function points which had been adjusted with 14 TCA factors and then used as input were inferior to neural networks using unadjusted function points with only three TCA factors being included as additional inputs in the neural network. If the TCA had modelled the technical complexity appropriately the use of the adjusted function point should not have been inferior. The recommendation therefore is that if the number of observations in the training set permit, that unadjusted function points be used together with all or the most significant TCA included as input.

A severe restriction on assessing the neural networks potential as an estimation tool was the sample size of the dataset. Even though the complexity factors and 21 development attribute factors had been recorded most were precluded from inclusion into the neural network model. As soon as more than four or five inputs were used the network included additional weights which caused the network to curve-fit.

The problem of the restricted sample size was aggravated in that the t-Test alone is not a reliable indicator of whether an attribute should be included into the network model or not. This was illustrated with the Mermaid dataset were the t-Test indicated that the difference in productivity between new and enhanced projects was not statistically significant. When this factor was introduced as an additional input into a neural network the reduction in MARE was significant.

If there are sufficient samples it is possible to include all factors which may influence the effort estimate and the neural network will eliminate those factors which are less relevant by adjusting its weight space appropriately. With a small sample and many inputs and weighted connections curve-fitting becomes a problem, which will then force the exclusion of some inputs. This may mean that some inputs which have a smaller influence on the goal value are not included and this will increase the prediction error.

7.2.6 Results Using the ASMA Project Database

The ASMA project dataset had the characteristic that some of the development attributes which had been recorded did not appear to have a significant effect on either software development productivity or effort. This could have resulted from several relevant project attributes not having been recorded. As these attributes may not be independent, but in some cases are likely to interrelated, that their effect on productivity was not identified. For example it is possible that in the presence of a specific attribute the second attribute may have a positive effect on effort, whereas the absence of the first attribute may cause the second attribute to have a negative effect on effort. If the first attribute is not recorded it is difficult to establish the effect of the second attribute.

The project dataset comprises a wide range of development environments resulting in considerable variability of development productivity. Other important factors are some missing data and other complicating issues such as four different time recording levels to record the effort expended on a project. All these contribute to making accurate effort estimation more difficult.

If necessary neural networks are capable of modelling complex relationships between the data inputs and the goal values. They are not capable of producing any relational functions if these are not inherent in the data. Some researchers have found [53] that system size contributes less than 50 percent of the variability of development effort. This figure may vary in different project databases, but as system size is not the only factor affecting development effort it would be unreasonable to expect neural networks to establish an accurate effort estimate from system size alone.

If the relationship between system size and effort does not exist to explain most of the variability then neural networks are not able to fabricate it to derive accurate estimates. In the ASMA trials the best results were obtained from the back-propagation networks. Using only function point size these estimated in the aggregated three test sets with a MARE of 0.29. If it had not been for one estimate with a very large error in Test Set 1, which may have been an outlier, the average error would have been even lower. Considering the amount of noise which is typically found in project data, together with a productivity variation of 34 times between the lowest and the highest project productivity, the effort estimate is considered to be reasonable.

With the inclusion of an additional six development attributes the MARE is reduced to an average of 0.17 on the aggregated three test sets. Again considering the factors mentioned above, and with still many other factors which may affect development effort not being included, this estimate is also considered to be satisfactory. On both criteria, the MARE and the percentage of estimates with an ARE of less than 0.25 the model meets the requirements set for an adequate estimation model.

In all the trials with the ASMA data the cascade networks did not perform to expectation. The k-NN algorithm trials indicated that this approach may hold some promise and may warrant further research effort. Even though the estimation accuracy did not match that of the back-propagation networks, it's estimation errors were

significantly lower than those of the cascade networks with whom they share a similar input file configuration.

7.2.7 Network Parameter Settings

The parameter settings of the cascade networks as discussed in Sections 3.6.3 and 4.3.1.5 produced good results in the research project trials. These settings were based on general experience with cascade networks [21] [25] [105] as well as the experience gained in informal preliminary testing on some of the data used in this thesis. Several different settings were tried but did not give consistently improved results.

In those cases where the back-propagation network models had a lower prediction error than cascade networks extensive trials were conducted using various parameter settings. Over-training may have been a source of the higher cascade prediction error [25]. To try and overcome this the output patience and candidate patience were reduced to attempt to stop the network from excessive training, but this did not significantly reduce the prediction error. The Output Decay, Mu and Epsilon, and the Candidate Input and Output Decay, MU and Epsilon were varied, but again this did not significantly reduce the prediction error. Reducing the Candidate and Output Change Threshold also did not solve the problem.

For the datasets which were used the weight range setting of 0.5 gave consistent good results. Of concern were the slightly inconsistent results of consecutive training runs. To attempt to increase the model stability without increasing the prediction error the weight range settings were reduced to 0.1, and the candidate pool was increased up to 100. The model performance did not give consistently improved results to warrant changing the initial parameter values.

The reasons for data scaling were discussed in Section 3.4. Experience with the pilot study (Section 4.6) indicated that logarithmic scaling resulted in improved prediction accuracy with the dataset which was used. This dataset was typical of project datasets and was characterised by large variations in prediction effort in which the network tended to overestimate the very low values and underestimate the very high values. The logarithmic compression facilitated a reduced prediction error. The various trials conducted with the cascade networks resulted in a similar experience. In general using

first a natural logarithm transformation followed by a scaling in the range of 0.1–0.9 for all continuous data gave consistently good results. All other inputs were coded as binary inputs with the values of either 0 or 1. Experiments with other values such as -1 and 1, -0.7 and 0.7, 0.3 and 0.7 did not reduce the prediction error.

The characteristics of different datasets vary and this may necessitate adjusting the parameter values accordingly for best neural network prediction performance. Several trials were conducted with each research project with different parameter settings to assess the appropriateness of the initially selected values. Generally the parameter settings were reasonably consistent throughout the datasets which were used and required minimal further adjustment.

7.3 Research Objective Assessment

The prime research objective was to answer the research question. The research question was:

Are artificial neural networks capable of providing an adequate development effort estimation model in which 75 percent of the predicted values are within 25 percent of the actual observations, and where the mean absolute relative error is less than 0.25?

The neural networks demonstrated their capability to extract a known function from a simulated equation dataset. They were also successful in accurately estimating well within the research question criteria the project effort in a large dataset of simulated project data which is likely to have contained considerably less noise than typically occurs in project data. This dataset met the requirements of sufficient observations for adequate training. The intuitive expectation is that estimation errors will increase as the level of noise in the dataset is increased.

The neural networks did not perform satisfactorily within the above criteria in the three small project datasets of Kemerer, Desharnis and Mermaid-2. These trials indicate the limitations of artificial neural networks under such extreme restrictions and their output is of limited use.

In the ASMA dataset the back-propagation networks demonstrated that they were able to estimate development effort within 25 percent of the actual effort in more than 75

percent of the projects in the test sets, and with a MARE of less than 0.25. Even this dataset did not fully meet the data requirements of neural networks as suggested by Baum and Haussler [10], and Hinton [44]. Due to the limited number of observations three randomly selected test sets were used. For an overall assessment the results of all test sets were aggregated.

Despite the restrictions of the project dataset, artificial neural networks have shown their ability to provide an adequate effort estimation model. With the limited number of observations and project attributes which were recorded the full potential of the estimation capability of neural networks could not be fully exploited and assessed. If the growth in projects in the ASMA dataset continues to increase at the current rate the database should within a few years provide a basis for the development of improved neural network estimation models. If in addition relevant project attributes are recorded this should further enhance the models which may be developed.

The conclusion thus is a conditional affirmation that the research objective was met, and that neural networks are capable of providing adequate estimation models. The performance of the neural network models is to a large degree dependent on the data on which they are trained, and the extent to which suitable project data is available will determine the extent to which neural networks can provide adequate effort estimation models. This limitation was demonstrated in very restricted datasets where the neural networks were not able to provide adequate estimation models. In contrast to this are the models trained and developed on the ASMA project data incorporating several development attributes which did meet the evaluation criteria set for an adequate estimation model.

7.4 Future Work and Recommendations

Neural network estimation models can be no better than the data on which they were trained and developed. The ASMA project dataset is a vital first step in providing such data. Neural networks and estimation models in general are no magic wands to generate something from nothing. Unless development attribute data on many of the relevant aspects of project development is recorded the accuracy of the estimation models will be limited. It is therefore recommended that consideration be given for

ASMA to take the initiative in including in their data capture form relevant development attributes.

The significant prior research indicated that inter-rater inconsistency may be as high as plus or minus 30 percent. The performance of the estimation models will be impeded by such discrepancies and accurate estimation will not be possible. Both ASMA and IFPUG have recognised this and the Function Point Counting Practices Manual Release 4.0 [35] and the International Software Benchmarking Standards Group version 1.0 collection package will contribute to more accurate measurement. It is recommended that research be conducted to ascertain to what extent inter-rater inconsistency has been reduced to improve the credibility of the function point count.

Despite these efforts function point counting remains a tedious and relatively costly operation. The perceived or actual unreliable function point count coupled with a high measurement cost detract from the widespread measurement of software projects. Reduced project measurement implies that project planning and control are more difficult, and the lack of data also impedes the metrics research effort. It is recommended that further research be directed toward improving the reliability of function point measures and reducing the cost of measurement. Some work has been done to suggest methods for automating this process, which should produce a consistent, reliable, and inexpensive project measure. These efforts need to be continued and the technique needs to be verified to enhance its credibility to lead to its widespread acceptance.

Either as a separate issue or as part of the process of reducing the measurement cost and increasing the reliability of the function point count it is suggested that some of the fundamental issues of function point counting be addressed. Initially the function point classification and coefficients were determined by trial and debate, and there has been no revision of these or the concept of their allocation since soon after their inception. The whole issue of process measurement which is largely ignored by function point analysis should also be addressed. The role of the function point technical complexity adjustment factors should be reviewed. Further research is warranted in determining their appropriateness and suggesting improvements if necessary.

The concept of the cascade neural networks has many advantages. In some of the trials in this research project, for some unknown reason the estimation error was greater than that of other neural network models. To exploit the full potential of the cascade network architecture and to produce consistent good estimates the precise cause of these problems should be identified and corrected.

To further improve the effort estimation process, especially when more reliable project data becomes freely available, research effort should be directed to investigating other estimation techniques such as case-based reasoning approaches and k-nearest-neighbour. These approaches, either alone or in combination with artificial neural networks should be investigated to facilitate improved effort estimation.

Appendix A

Function Point Analysis

Introduction

Function Points (FPs) were originally suggested in 1979 by Albrecht [1] of IBM. The original methodology was revised in 1984 and has undergone continuous refinement and with clearer and more precise definitions. In 1984 IBM started to include courses in Function Point Analysis (FPA) as part of its data processing curriculum, which created a quantum leap in overall utilisation of the technique [56]. With IBM's support and largely from IBM users, the International Function Point Users Group (IFPUG) was formed with the aim to promote and standardise the use of FPs. Other approaches such as Symons FPA MkII method and also DeMarco's "Bang" metric did not have such organisational backing and lack the popularity of the IBM FP method [56].

Since the initial FP proposals in 1979 there have been numerous alterations. With the formation of IFPUG, definitions have been continuously refined and have become the popular standard.

A FP count of a system measures two components. These are firstly the information processing size, expressed in unadjusted FPs (UFPs), and secondly the technical complexity measure, expressed as the Technical Complexity Adjustment (TCA).

The Information Processing Size

The information processing size is determined by categorising a system into 5 classifications. These are external inputs, outputs, inquiries, interfaces to other systems, and internal files and are described in more detail below[56].

External inputs are screens or forms through which the user adds new data, or updates existing data. If a screen, which typically is 80 columns by 25 lines, is too large for a single normal display, and flows over onto a second screen it is counted as a single input. Inputs that require unique processing are the determinant factor.

External outputs are screens or reports which the application produces. As with inputs, outputs requiring separate processing are the unit of measure. For example in an application for a university, a student enrolment report with say 10 000 students, would be counted as one output.

Inquiries are screens which allow users to interrogate the system. Assistance or information such as 'HELP' screens would fall into this category.

External interface files are those shared with other applications, such as incoming or outgoing tape files, shared databases, and parameter lists.

Internal files are the data files or logical collections of records which the application modifies or updates. These include floppy disk files, magnetic tape files, flat files, a leg in a hierarchical database, a table in a relational database, and a path through a net in a network-oriented database.

Each of these five categories is further classified as either low, average, or high, depending on the number of data elements in each type, and other factors. Tables A.1 through A.6 show the criteria to determine the final classification for inputs, outputs, inquiry input portions, inquiry output portions, external interface files, and logical internal files respectively.

Table A.1 External Inputs

	Data element types		
Files Referenced	**1-4**	**5-15**	**>15**
0-1	low	low	average
2	low	average	high
>2	average	high	high

Table A.2 External Outputs

	Data element types		
Files Referenced	**1-5**	**6-19**	**>19**
0-1	low	low	average
2-3	low	average	high
>3	average	high	high

Table A.3 Inquiry Inputs

	Data element types		
Files Referenced	**1-4**	**5-15**	**>15**
0-1	low	low	average
2	low	average	high
>2	average	high	high

Table A.4 Inquiry Outputs

	Data element types		
Files Referenced	**1-5**	**6-19**	**>19**
0-1	low	low	average
2-3	low	average	high
>3	average	high	high

Table A.5 Interface Files

	Data element types		
Files Referenced	**1-19**	**20-50**	**>50**
0-1	low	low	average
2-5	low	average	high
>5	average	high	high

Table A.6 Logical Internal Files

	Data element types		
Files Referenced	**1-19**	**20-50**	**>50**
0-1	low	low	average
2-5	low	average	high
>5	average	high	high

Each classification is allocated a number of points or weights, and the sum for all components is expressed as the number of unadjusted function points. Table A.7 shows the matrix of the points allocation for each classification. The external inquiry classification remains a single category by grouping the inquiry input count (Table A.3) and the inquiry output count (Table A.4).

This step of classification and weight allocation at first looks deceptively simple but in real-life projects is time-consuming and the source of some inconsistency.

Table A.7 Function Point Allocation

Description	Simple	Average	Complex
External Input	3	4	6
External Output	4	5	7
External Inquiry	3	4	6
External Interface File	5	7	10
Internal File	7	10	15

The Technical Complexity Adjustment (TCA)

The treatment of complexity is somewhat subjective, but its determination is supported by guidelines for interpretation.

The TCA is derived by the following equation:

$$\mathbf{TCA = 0.65 + 0.01 \times DI}$$

where

DI is the degree of influence.

The degree of influence is determined by 14 influential complexity factors. Each of these is evaluated on a scale of 1 to 5, depending on the degree to which the factor is present in the system. A 0 would be allocated if the factor was absent, while a 5 would be allocated if the factor exerted a strong influence throughout the system. The other values fall between these two extremes (decimal values are permitted), depending on their degree of influence. The sum of all 14 factors determined in this way is termed the Degree of Influence (DI).

The 14 influential factors, C1 through C14, are the following and points or degrees of influence are allocated as follows:

C1 **Data communication**: This implies that data and/or control information would be sent or received over communication facilities.

0 Batch applications

1 Remote printing or data entry

2 Remote printing and data entry

3 A teleprocessing front end to the application

4 Application with significant teleprocessing

5 Application that is predominantly teleprocessing

C2 **Distributed functions:** Their score is determined by whether an application is monolithic and operates on a single contiguous processor or is distributed among a variety of processes.

0 Pure monolithic application

1 Application that prepares data for other components

2 Application distributed over a few components

3 Application distributed over more components

4 Application distributed over many components

5 Application dynamically performed on many components

C3 **Performance objectives:** A score of 0 is allocated if no special performance criteria are stated in the requirements specification. A score of 5 is allocated if the users insist on very stringent performance targets that require considerable effort to achieve.

C4 **Heavily used configuration:** If the application has no special usage constraints 0 is scored. If the anticipated usage requires considerable special effort to achieve, a 5 is scored.

C5 **Transaction rate:** If the volume of transactions is not significant the score is 0, while a 5 is scored if the volume of transactions is high enough to stress the application and require special effort to achieve desired throughputs

C6 **On-line data entry:** A 0 is scored if none or fewer than 15 percent of the transactions are interactive. A 5 is scored if more than 50 percent of the transactions are interactive.

C7 **End-user efficiency:** This is scored 0 if there are no end-users or there are no special requirements for end-users. If stated requirements for end-users are stringent enough to require special circumstances to achieve them a 5 is scored.

C8 **On-line data update:** If there is none, 0 is allocated to this category, and 5 if the on-line updates are both mandatory and especially difficult, possibly because of the need to back-up or protect data against accidental change.

C9 **Complex processing:** This is scored 0 if there is no complex processing and 5 in cases requiring extensive logical decisions, complicated mathematics, tricky exception processing, or elaborate security schemes.

C10 **Re-useability:** If the functionality is planned to stay local to the current application a 0 is scored. If much of the functionality and the project deliverables are intended for widespread utilisation by other applications a 5 is scored.

C11 **Installation ease:** This is allocated a score of 0 if this factor is insignificant, and 5 if the installation is both important and so stringent that it requires special effort to accomplish a satisfactory installation.

C12 **Operational ease:** This is allocated a score of 0 if the provision of this factor is insignificant and 5 if operational ease is so important that it requires special effort to achieve it.

C13 **Multiple sites:** A 0 is scored if there is only one planned location and a 5 if the project and its deliverables are intended for many diverse locations

C14 **Facilitate change:** If change does not occur a 0 is scored, and a 5 if the application is developed specially to allow end-users to make rapid changes to control data and tables.

The 14 individual scores are summed, multiplied by 0.01, and a constant of 0.65 is then added, to derive the total complexity adjustment (TCA) factor. Each degree of influence is thus worth approximately one percent of the TCA. From the equation it can be seen that the TCA can yield a value which can range from 0.65, if none of the 14 influential factors are present, to 1.35 if all of them are exhibit a strong influence. This implies that the unadjusted function point value adjustment can range from a reduction of 35 percent through to an increase of 35 percent to derive the function point count.

FP Counting Practice

The current IBM function point methodology is substantially more rigorous than the original 1979 implementation. This rigour has added significantly to the effort required to derive the function point count. IFPUG (International Function Point Users Group) has developed a manual which defines the function point counting convention in an effort to standardise function point counts. Release 4.0 was made available in January 1994. The manual is comprehensive and lengthy, and reinforces the suggestion that function point counting should only be done by trained function point personnel. The effort involved to count function points of a large system using the current IBM methodology amounts to several days [56], thus implying a significant cost.

Appendix B

Back-Propagation Networks

Introduction

In this section some basic back-propagation neural network concepts are included to give some background information on artificial neural networks. Some of the concepts, such as for example the neuron structure, are common to other neural network architectures such as cascade networks as well.

Neurons

Originally neural networks were aimed towards modelling networks of real neurons in the human brain. The models are extremely simplified when seen from a neurophysiological point of view, although they are still considered valuable for gaining insight into the principles of biological computation.

The human brain is composed of approximately 10^{11} (100 billion) neurons, each connected to as many as 1000 others [43]. A neuron operates by receiving signals from other neurons through connections called synapses. The combination of these signals, in excess of a certain threshold, will result in the neuron firing. The neuron firing results in sending a signal on to other neurons connected to it. Some signals act as excitations and others as inhibitions to a neuron firing. This massive number of neurons and their complex interconnections, with the presence or absence of firings in the pattern of synaptic connections results in the human brain's thinking ability.

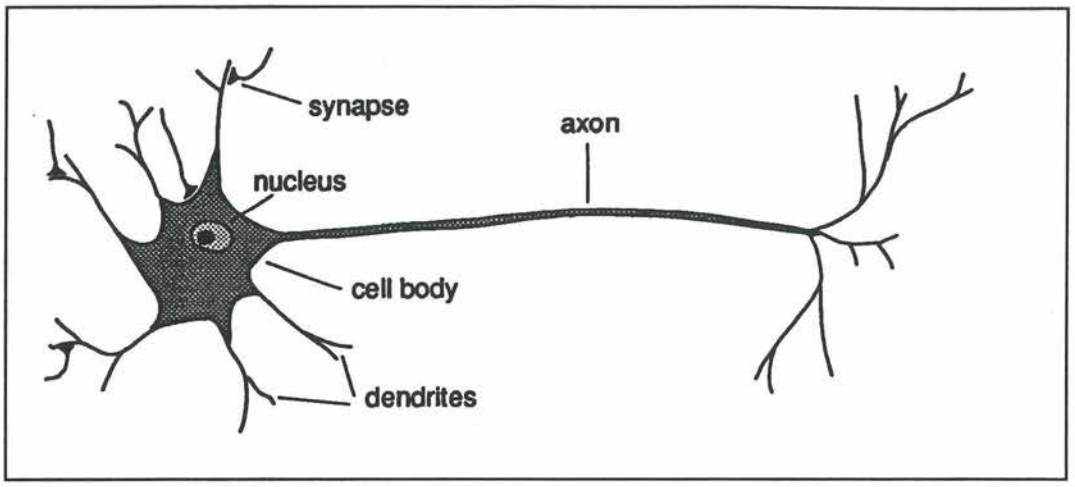


Figure B.1 Structure of a Human Neuron [43]

The structure of a human neuron is shown in Figure B.1 and the Neuralyst manual [80] describes the neurophysiological process as follows:

'Each neuron has a body called the soma. The soma is much like the body of any other cell in that it contains the cell nucleus, various bio-chemical factories and other components that support ongoing activity. Surrounding the soma are dendrites which are the receptors for signals generated by other neurons. These signals may be either excitatory or inhibitory. All signals at the dendrites of a neuron are combined and the result will determine whether that neuron will fire. If the neuron fires, an electrical impulse is generated. This impulse starts at the base, called the hillock, of a long cellular extension, called the axon, and proceeds down the axon to its ends, called butons.'

According to Hertz, Krogh and Palmer [43], at the end of these butons are the transmitting ends, the synapses. When a neuron fires the electrical impulse stimulates the buton resulting in electrochemical activity, which transmits the signal across the synaptic gap to the receiving neuron.

When the neuron is at rest it maintains an electrical potential of approximately 40–60 millivolts. When the neuron fires it creates an electrical impulse of approximately 90–110 millivolts, which travel at 0.5–100 metres per second and lasts for about a

millisecond [80]. After firing the neuron has to rest for several milliseconds, called the refractory period, before it can fire again [43]. In some instances neurons may fire at the rate of 100 times per second.

When this is compared to a fast electronic computer, whose signals travel at about 2×10^8 metres per second, which is approximately two thirds that of light in free air, and whose impulses last for 10 nanoseconds, requiring no refractory period, implying that impulses can succeed each other continuously, it has at least a 2,000,000 times advantage in signal transmission and a 1,000,000 times advantage in the signal repetition rate [80].

Judged on signal speed and transmission rate alone the computer is clearly superior in processing performance. What the human brain lacks in speed it more than makes up for in the number of neurons and the interconnection complexity between these elements, and this manifests itself in that it is not as quick at arithmetic, but is many times faster and very much more capable at pattern recognition and the perception of relationships. Another aspect in which the brain is hugely superior to current artificial neural networks is its capability to 'self-program' itself or learn by adapting in response to changing external stimuli.

Artificial neural networks are models of the biological structures and are an attempt to exploit some of the human brain's potential. This research project investigated the potential of using back-propagation and cascade network models to estimate effort in the complex software development environment.

Artificial Neuron Structure

The artificial neural networks are modelled on the biological structures. The typical artificial neuron consists of multiple inputs, and a single output. This is illustrated in Figure B.2, where the inputs are marked x_1 to x_i where i is the number of inputs to that neuron j.

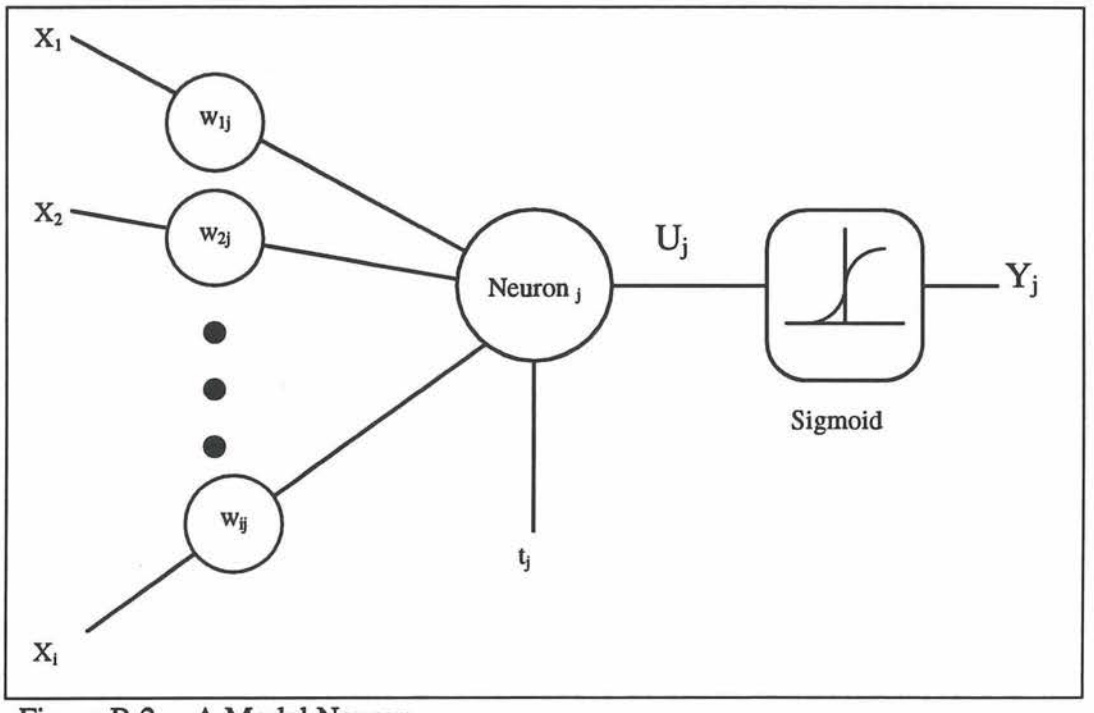


Figure B.2 A Model Neuron

Neuron Operation

The back-propagation network operations are described here as it is the most prevalent and generalised network currently in use [80]. There is a fair understanding of how neurons work, but there is some uncertainty regarding the manner in which neurons organise themselves and the mechanisms used by arrays of neurons to adapt their behaviour to external stimuli. This topic is the focus of considerable research effort [92] [93], and of particular interest is the extraction of knowledge from the weight space in which the network has captured a representation of the problem domain [22] [101].

Each input into the neuron is modified by multiplying each input by a weight for that connection. In Figure B.2 these are marked as w_{1j} through to w_{ij} for all the inputs. These weighted inputs are then summed by the neuron, and with reference to a threshold value, will determine the neuron output. The output is described by two sets of equations. The first one, which is the combination operation of the neuron yielding U_j for the jth neuron [44], is:

$$U_j = \sum(x_i \times w_{ij}) - t_j$$

U_j is biased adjusted by a previously established threshold value, t_j and is then subjected to the activation or threshold function. Where the activation function is sigmoidal, the equation is as follows [44]:

$$Y_j = (1 + e^{-U_j})^{-1}$$

This equation implements the firing of the neuron. Numerous activation functions have been used, such as gaussian, linear, step-wise, or the most commonly used one which is the sigmoid function. A graphical display of the sigmoidal function output mapping is shown in Figure B.3. This output, Y_j, which is between zero and one, will be the input to the next layer, or it will be the response of the network if this neuron is in the last layer.

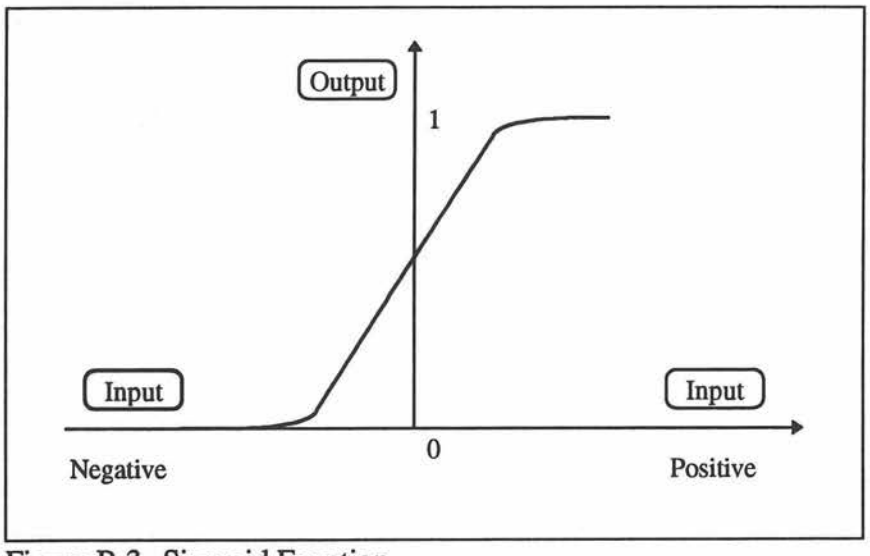


Figure B.3 Sigmoid Function

Back-Propagation Network Topology

Neurons are typically arranged in layers. A layer has all its inputs connected to either a preceding layer, or to the inputs from the external world. Neurons within a layer are not directly connected to each other. Multiple layers are arranged in the following manner, which is depicted graphically in Figure B.4. There is an input layer, succeeded by zero or more intermediate or hidden layers, and finally an output layer. As the

intermediate layers have no direct inputs or outputs to the external world they are known as the hidden layers.

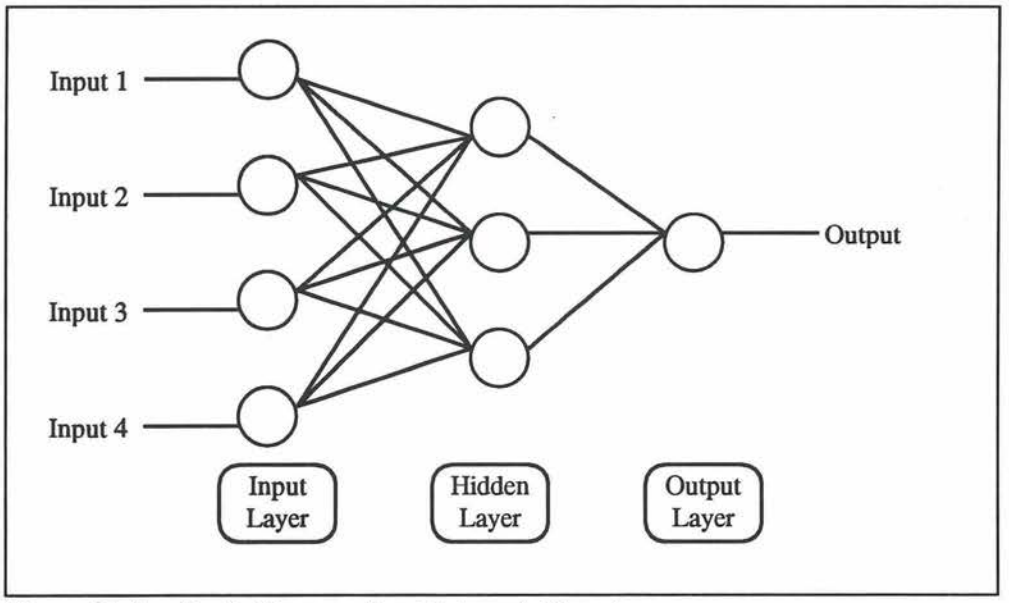


Figure B.4 Back-Propagation Network Structure

Back-propagation layers are usually fully connected. This means that each neuron in the input layer has a single input, and is connected to every neuron in the first hidden layer, or the output layer if there is no hidden layer. Each neuron in the hidden layer is connected to every output from the preceding layer, which could be either an input layer or another hidden layer, as well as being connected to every neuron in the succeeding layer, which in this case could be either another hidden layer or the output layer. The output layer neuron has a connection to every neuron in the preceding layer, and a single output to the external world [80].

The advantages of hidden units is that as more are added to the network they are able to extract progressively more complex features from the inputs [44]. This allows for more complex tasks to be learned. The disadvantage of hidden units is that learning is much harder. The learning procedure must implicitly decide what features should be represented by the hidden units. Learning now has to be achieved by searching a much larger space of possible functions, as the learning procedure must implicitly choose between alternative representational schemes [44].

Some Applications of Back-Propagation Networks

The recent growth in back-propagation network applications has been quite remarkable. In the last five years many industrial and commercial applications have been developed. The details of many are considered corporate property and are shrouded in secrecy. Some well known back-propagation network applications are [44]:

- Interpreting medical images
- Diagnosing cardiac arrest in emergency rooms
- Detecting bombs in suitcases
- Controlling various chemical processes
- Predicting currency exchange rates
- Diagnosing failures in life support machines
- Dynamically distorting telescope mirrors to cancel out the atmospheric distortion
- Steering autonomous land vehicles
- Military use - devices for destroying targets

The following examples are included to illustrate some back-propagation network applications and are all taken from Hinton [44].

SAIC Bomb Detector

The SAIC system was developed for detecting bombs in suitcases. Sensors are used to irradiate luggage with neutrons. The emitted gamma-rays are detected by a sensor array. The energy of the gamma-rays indicates the elements present. Bombs typically have high nitrogen concentrations. The input to the network is 20 values derived from 200 sensor values which are normalised and lie between -0.5 and +0.5. The network is a feed forward net with one hidden layer and three units in the output layer, to predict the presence or absence of lump explosives, sheet explosives, or either. The network typically is required to be trained for 2000 epochs. The performance of the network outperforms discriminant analysis and is now commonly used at airports. Depending on the threshold value chosen the network can predict the presence of an explosive

device within various probability ranges of false alarms. For example with a three percent probability of a false alarm, the probability of detection is approximately 95 percent. With the threshold value set lower, the probability of a false alarm rises to 10 percent, while the probability of detection increases to approximately 98.5 percent.

Autonomous Land Vehicle System

With the Pomerleau system which steers autonomous land vehicles, only a simple back-propagation network performs as well as a conventional artificial intelligence approach which took many person-years to develop. Once the network has been trained it runs eight times as fast as the artificial intelligence system. The input to this network is extracted from a 30×32 video input retina, and a 8×32 range finder input retina. The single hidden layer has 29 units, while the output layer has 45 units, to indicate the various steering positions, ranging from sharp left, to straight ahead, to sharp right. The network trains quickly with the vehicle only having to be driven for a short time. The reliability of the system is good as long as the road conditions are similar to the test data. For example when the vehicle was trained on a training circuit, it became confused when it was used on the open road and experienced oncoming traffic which it had not encountered before.

Pap Smear Evaluation

One of the most valuable applications of back-propagation neural networks has been in the interpretation of pap smears for cervical cancer (Pap-Net). Every year approximately 50 million pap smears are submitted for analysis. Each pap smear has approximately 500,000 cells. On average about 97 percent of the smears are clear. A bad smear may have as few as 10 cells with enlarged nuclei when the DNA behaves abnormally. Typically it takes a cyto-technician a long time to examine a slide, and the failure rate to detect bad smears is approximately 30 percent. One reason for this is that human pattern recognition has great difficulty because cells overlap and they contain random junk as well.

With the Pap-net system, fast conventional image processing hardware is used to select 50,000 cells (10 percent) to examine. A back-propagation network is used to assign a

suspicious score to a roughly centred image of a cell, or some overlapping cells. The network will identify and display the 128 most suspicious cells, together with their contexts, for a technician to examine. The details of Pap-Net are confidential.

The combined system is much faster and less stressful for the technicians, and the miss rate is only approximately three percent.

Haykin's Iceberg Detector

Back-propagation networks have proved a good predictive model for normal cases. This implies that an alert can be generated whenever the model's predictions of the normal cases are badly wrong. This is used typically for monitoring heart patients, and chemical plants. Another useful such example is Haykin's iceberg detector. Small icebergs or growlers are difficult to detect in radar returns, as the waves give strong background clatter. This problem is solved by modeling this chaotic clutter and using a neural network to predict the next radar return from the previous seven returns.

The predictive model adapts to the ocean conditions as these change with time. Patches where the prediction fails may contain icebergs, as the wave pattern around them is distorted. Haykin's iceberg detector has given good results.

Conclusion

These examples illustrate the powerful pattern recognition and modelling ability when fairly simple neurons as were discussed previously, are combined and trained in a neural network structure.

Appendix C

Sundry Project Information

Mermaid-2 Productivity Adjustment Factors

The Mermaid-2 project data recorded 21 productivity adjustment factors which are listed below.

1. User involvement
2. User commitment
3. User experience with the application
4. Staff turnover
5. Computer resource availability
6. System response time
7. Time constraints
8. Staff constraints
9. Experience of team
10. Requirements stability
11. System familiarity
12. Problem complexity
13. Complexity of user interface
14. Structured methods

15. Familiarity with structured methods
16. Tools/software usage
17. Team experience with tools
18. Programming language level
19. Familiarity with programming language
20. Project management experience of project leader
21. Working environment

Appendix D

Data

This appendix includes some results which were not included in the main thesis. The prediction results of the equation simulation are given in Table D.1 To test the stability of Cascade2's dynamic network generation, the weights were extracted and examined. The weight values for the two networks examined are given in Table D.2.

The effort estimation results from the simulated project dataset are shown in Table D.3, while the output and results of the back-propgation neural network model developed on the ASMA project dataset are shown in Table D.4

Table D.1 Prediction Results of Equation Simulation

Testing Set	Function Points	Average Team Size	Actual Productivity	Estimated Productivity	Relative Error
1	100	1	1.367	1.325	-0.030
2	100	2	1.164	1.152	-0.010
3	100	3	0.927	0.925	-0.002
4	100	4	0.726	0.712	-0.019
5	100	5	0.565	0.558	-0.012
6	100	6	0.438	0.443	0.011
7	100	7	0.339	0.304	-0.104
8	100	8	0.262	0.254	-0.030
9	100	9	0.202	0.200	-0.013
10	100	10	0.156	0.154	-0.012
11	100	11	0.121	0.132	0.095
12	100	12	0.093	0.107	0.153
13	100	13	0.072	0.084	0.171
14	100	14	0.055	0.064	0.164
15	100	15	0.043	0.048	0.125
16	200	1	1.257	1.244	-0.010
17	200	2	1.183	1.169	-0.011
18	200	3	0.974	0.998	0.025
19	200	4	0.776	0.752	-0.031
20	200	5	0.610	0.607	-0.004
21	200	6	0.476	0.492	0.035
22	200	7	0.370	0.406	0.098
23	200	8	0.287	0.300	0.043
24	200	9	0.222	0.229	0.030
25	200	10	0.172	0.176	0.023
26	200	11	0.133	0.134	0.007
27	200	12	0.103	0.110	0.065
28	200	13	0.079	0.083	0.048
29	200	14	0.061	0.060	-0.016
30	200	15	0.047	0.036	-0.239
31	400	1	0.946	0.992	0.048
32	400	2	1.088	1.089	0.001
33	400	3	0.958	0.970	0.012
34	400	4	0.789	0.776	-0.016
35	400	5	0.632	0.630	-0.004
36	400	6	0.500	0.514	0.027
37	400	7	0.393	0.409	0.040
38	400	8	0.307	0.294	-0.041
39	400	9	0.239	0.234	-0.021
40	400	10	0.186	0.189	0.019

Table D.1 Prediction Results of Equation Simulation (continued)

Testing Set	Function Points	Average Team Size	Actual Productivity	Estimated Productivity	Relative Error
41	400	11	0.144	0.139	-0.038
42	400	12	0.112	0.112	0.004
43	400	13	0.086	0.091	0.051
44	400	14	0.067	0.065	-0.029
45	400	15	0.052	0.047	-0.084
46	1000	1	0.332	0.296	-0.109
47	1000	2	0.696	0.686	-0.014
48	1000	3	0.748	0.761	0.017
49	1000	4	0.681	0.688	0.011
50	1000	5	0.580	0.578	-0.003
51	1000	6	0.478	0.479	0.003
52	1000	7	0.386	0.381	-0.013
53	1000	8	0.308	0.307	-0.003
54	1000	9	0.244	0.247	0.011
55	1000	10	0.192	0.192	0.001
56	1000	11	0.151	0.150	-0.002
57	1000	12	0.118	0.118	0.000
58	1000	13	0.092	0.093	0.013
59	1000	14	0.072	0.075	0.040
60	1000	15	0.056	0.056	0.005
61	2000	1	0.050	0.046	-0.091
62	2000	2	0.287	0.271	-0.056
63	2000	3	0.431	0.440	0.021
64	2000	4	0.464	0.462	-0.005
65	2000	5	0.437	0.434	-0.007
66	2000	6	0.384	0.379	-0.013
67	2000	7	0.326	0.327	0.005
68	2000	8	0.269	0.270	0.003
69	2000	9	0.219	0.215	-0.019
70	2000	10	0.177	0.180	0.017
71	2000	11	0.141	0.137	-0.032
72	2000	12	0.112	0.112	0.001
73	2000	13	0.089	0.089	0.005
74	2000	14	0.070	0.072	0.025
75	2000	15	0.055	0.057	0.046
76	3000	1	0.007	0.010	0.345
77	3000	2	0.113	0.115	0.015
78	3000	3	0.237	0.231	-0.023
79	3000	4	0.301	0.296	-0.018
80	3000	5	0.313	0.315	0.006

Table D.1 Prediction Results of Equation Simulation (continued)

Testing Set	Function Points	Average Team Size	Actual Productivity	Estimated Productivity	Relative Error
81	3000	6	0.295	0.295	0.000
82	3000	7	0.262	0.266	0.014
83	3000	8	0.225	0.224	-0.001
84	3000	9	0.188	0.186	-0.011
85	3000	10	0.155	0.152	-0.021
86	3000	11	0.126	0.128	0.014
87	3000	12	0.102	0.103	0.009
88	3000	13	0.081	0.088	0.087
89	3000	14	0.065	0.064	-0.008
90	3000	15	0.051	0.052	0.012

Table D.2 Weights of Comparative Networks

Weight No	Network 1	Network 2
1	0.2493	0.2683
2	-0.2268	-0.2249
3	-0.2434	-0.2254
4	-0.2421	-0.2309
5	-0.1823	-0.1539
6	-0.1933	-0.1902
7	0.1887	0.1818
8	0.2067	0.1712
9	0.2067	0.2018
10	0.2032	0.1886
11	0.1876	0.1727
12	0.1368	0.1230
13	0.1587	0.1204
14	0.1546	0.1579
15	0.2091	0.1770
16	0.1546	0.1591
17	0.0514	0.0441
18	0.0411	0.0327
19	0.0640	0.0636
20	0.0684	0.0656
21	0.0726	0.0632

Table D.2 Weights of Comparative Networks (continued)

Weight No	Network 1	Network 2
22	-0.1877	-0.1756
23	-0.1936	-0.2030
24	-0.1618	-0.1554
25	-0.1500	-0.1136
26	-0.1423	-0.1096
27	0.0638	0.0386
28	0.0660	0.5627
29	0.0841	0.6842
30	0.0931	0.0785
31	0.0928	0.0968
32	-0.0341	-0.0051
33	-0.0245	-0.0387
34	-0.0487	-0.0198
35	-0.0271	-0.0362
36	-0.0155	0.0080
37	0.2408	0.2473
38	0.2357	0.2182
39	0.2473	0.2475
40	0.2814	0.0245
41	0.2860	0.2664
42	-0.3607	-0.3416
43	0.6597	0.6351
44	-0.1217	-0.0906
45	-0.5488	0.4671
46	-0.0289	-0.0340
47	0.0459	-0.0145
48	0.0312	-0.0237
49	-0.0130	0.0258
50	0.0082	-0.0137
51	-0.0243	0.0137
52	0.0394	0.0144
53	0.0372	-0.0079
54	-0.0231	0.0149
55	-0.0482	-0.0302
56	-0.0220	-0.0133
57	0.0102	-0.0170
58	-0.0100	

Table D.3 Results of Effort Estimation from Simulated Dataset

Test	Network Inputs									SPQR	NN	
No	1	2	3	4	5	6	7	8	FP	Effort	Estimate	ARE
1	3	4	2	3	2	4	5	3	220	6.5	6.2	0.042
2	1	4	3	4	3	2	1	4	5100	210.2	233.2	0.110
3	2	1	2	4	1	2	3	2	5133	128.5	118.0	0.082
4	2	2	3	5	5	4	2	2	3711	235.3	223.9	0.048
5	2	4	4	4	2	1	3	4	971	23.0	24.3	0.054
6	2	4	3	2	4	5	3	3	1435	87.2	87.5	0.004
7	4	5	4	4	2	2	3	5	690	26.9	26.6	0.011
8	3	2	5	2	2	4	3	4	348	9.2	9.1	0.008
9	3	2	3	2	2	2	4	2	676	14.7	15.2	0.036
10	3	2	3	1	4	3	2	2	2133	96.5	96.9	0.004
11	4	3	2	3	3	1	5	2	3235	162.6	169.3	0.041
12	1	2	5	3	3	3	1	4	3265	113.1	112.4	0.006
13	1	2	1	4	5	1	4	2	5067	189.2	181.2	0.042
14	1	2	5	3	1	4	5	1	2619	90.9	90.2	0.007
15	5	1	5	4	5	3	4	1	676	32.7	35.7	0.092
16	1	3	3	1	3	2	1	4	967	27.5	28.4	0.034
17	3	4	1	4	3	1	5	2	2706	96.2	102.5	0.066
18	2	2	5	2	3	2	3	5	3000	118.7	115.2	0.029
19	5	4	3	4	2	4	2	2	1676	78.8	79.1	0.004
20	1	3	4	1	1	1	1	3	6409	164.9	172.8	0.048
21	5	1	1	5	3	1	4	5	1095	44.7	43.6	0.023
22	4	5	4	4	2	1	2	2	4864	217.7	216.3	0.006
23	3	3	1	4	1	3	3	3	2316	73.3	68.5	0.065
24	3	1	2	5	5	5	1	2	2588	123.7	129.3	0.045
25	3	2	3	1	1	5	2	3	2167	84.7	91.8	0.083
26	3	4	4	5	4	3	1	2	3423	173.7	174.0	0.001
27	5	1	5	3	2	1	2	5	2871	122.4	123.3	0.007
28	5	3	4	4	2	1	1	1	7500	339.2	303.9	0.104
29	1	5	2	5	2	2	3	5	1619	44.9	44.4	0.010
30	3	3	2	2	1	2	3	4	2684	81.4	78.0	0.042
31	5	2	2	5	3	3	1	3	3933	213.2	211.6	0.008
32	1	1	3	4	2	5	4	4	2463	173.0	159.6	0.077
33	3	5	5	5	1	3	2	3	1794	51.1	56.1	0.098

Table D.3 Results of Effort Estimation from Simulated Dataset (continued)

Test	Network Inputs									SPQR	NN	
No	1	2	3	4	5	6	7	8	FP	Effort	Estimate	ARE
34	1	5	4	3	5	1	3	2	3308	152.4	139.8	0.082
35	1	5	2	1	3	5	3	4	1580	76.4	79.7	0.043
36	4	3	5	3	4	4	2	5	2913	293.6	314.4	0.071
37	4	3	5	1	1	3	4	4	1804	87.3	81.7	0.065
38	1	4	5	2	1	2	3	2	4867	119.5	130.7	0.094
39	2	3	3	5	2	2	1	1	2833	78.6	80.5	0.025
40	1	3	4	4	1	5	3	4	1280	52.3	56.5	0.080
41	4	3	1	2	5	1	1	1	5143	281.3	274.0	0.026
42	1	1	5	2	2	1	1	3	818	13.1	12.0	0.084
43	3	4	1	3	2	2	2	4	265	4.2	4.5	0.076
44	2	2	2	2	4	5	5	2	300	12.7	12.3	0.036
45	5	4	5	1	2	2	5	1	2118	85.5	84.7	0.010
46	1	5	2	2	4	3	3	4	905	32.4	32.3	0.002
47	2	3	2	2	5	4	1	2	4967	225.6	250.2	0.109
48	5	5	1	4	1	4	4	5	2852	223.9	213.0	0.049
49	2	4	2	3	3	2	1	3	4231	147.8	147.9	0.000
50	2	1	4	5	4	5	2	2	1789	93.8	101.2	0.078
51	5	3	3	5	1	4	1	4	1368	67.5	68.2	0.009
52	4	3	2	1	1	5	4	1	3524	185.2	189.3	0.022
53	5	1	3	2	4	5	2	2	3579	342.2	324.5	0.052
54	3	5	5	3	2	4	2	1	233	3.7	3.9	0.043
55	1	5	1	1	3	2	2	4	2324	74.4	72.6	0.024
56	5	2	3	5	5	5	1	1	700	40.9	39.9	0.026
57	2	5	5	2	3	4	1	4	2947	137.6	122.8	0.108
58	1	1	3	3	5	2	5	5	1260	67.1	70.8	0.055
59	1	2	1	4	1	1	4	3	3618	82.6	82.5	0.001
60	1	2	5	3	4	3	2	5	3048	161.8	150.3	0.072
61	4	5	4	4	4	2	4	1	4533	352.6	345.5	0.020
62	1	3	2	5	4	2	5	2	3000	129.0	127.9	0.008
63	5	2	4	2	4	2	3	1	5615	412.8	453.9	0.099
64	1	1	4	5	4	5	4	5	690	56.3	51.5	0.085
65	2	4	1	4	5	3	4	2	1079	43.5	43.3	0.007
66	1	3	2	2	2	1	1	3	6364	156.0	171.8	0.101
67	5	4	3	3	4	5	4	1	1143	98.7	97.8	0.009

Table D.3 Results of Effort Estimation from Simulated Dataset (continued)

Test	Network Inputs									SPQR	NN	
No	1	2	3	4	5	6	7	8	FP	Effort	Estimate	ARE
68	3	4	4	2	1	2	4	3	2947	102.0	108.0	0.059
69	4	1	5	1	1	2	5	3	2595	106.9	104.0	0.027
70	3	4	1	3	2	4	2	1	400	7.0	7.2	0.028
71	1	5	4	1	5	3	4	3	3262	207.7	203.3	0.021
72	4	1	2	4	4	2	4	4	1310	74.9	76.7	0.024
73	4	2	4	2	4	2	2	5	368	17.4	16.6	0.046
74	3	5	5	2	2	3	5	5	1741	82.9	78.7	0.050
75	5	4	1	2	1	5	3	5	2741	199.1	182.6	0.083
76	1	4	1	1	5	3	2	5	1048	42.3	39.3	0.071
77	3	3	1	1	3	2	4	4	119	2.4	2.5	0.047
78	3	5	1	2	1	5	1	3	2526	84.8	84.1	0.009
79	4	4	5	5	3	5	1	1	300	9.6	9.7	0.010
80	4	4	3	2	5	2	2	2	5192	416.3	395.5	0.050
81	1	3	4	2	5	5	1	1	2967	144.4	151.3	0.048
82	5	4	2	2	4	1	3	2	5346	332.0	314.8	0.052
83	5	3	1	4	5	2	2	4	1441	81.7	75.2	0.080
84	2	1	5	2	3	2	3	3	1029	28.3	29.4	0.039
85	2	4	3	3	4	2	2	3	1233	49.7	45.7	0.080
86	2	5	4	1	2	5	5	3	815	40.4	37.8	0.065
87	4	1	4	4	5	5	2	4	1391	140.4	130.7	0.069
88	3	3	2	1	3	4	2	1	3733	140.3	133.7	0.047
89	5	4	5	2	2	3	5	3	1935	104.8	104.4	0.004
90	5	1	4	1	4	2	4	3	1579	108.9	103.1	0.053
91	3	2	2	2	4	3	5	1	1868	80.8	75.5	0.065
92	3	2	4	2	1	1	1	5	1033	24.5	22.8	0.070
93	1	3	2	2	4	3	5	2	1738	76.3	80.5	0.054
94	5	5	5	1	1	1	2	5	4294	183.8	193.5	0.053
95	5	2	4	3	3	1	1	1	6643	385.8	395.8	0.026
96	3	5	2	5	5	2	5	2	2974	150.7	153.5	0.019
97	1	2	5	5	4	5	1	2	794	24.7	22.7	0.081
98	4	4	2	4	1	5	1	2	2000	77.9	74.7	0.041
99	4	3	2	4	4	1	2	4	2300	137.7	140.1	0.017
100	4	2	5	3	4	3	1	3	1367	75.4	79.5	0.054

Table D.4 ASMA Back-Propagation Estimates

Estimate	Actual	ARE
190	247	0.23
1267	1178	0.08
232	286	0.19
1962	2506	0.22
241	274	0.12
623	539	0.16
5413	6817	0.21
2443	2596	0.06
10944	14716	0.26
5168	5369	0.04
795	704	0.13
11648	14716	0.21
2601	2567	0.01
1814	2526	0.28
2614	3251	0.20
2455	4183	0.41
751	748	0.00
6630	7453	0.11
15693	17245	0.09
1214	1544	0.21
2462	2386	0.03
4363	8080	0.46
762	728	0.05
10182	10184	0.00
7646	7453	0.03
5847	4292	0.36
1101	2236	0.51
3758	4095	0.08
21996	21491	0.02
171	279	0.39
MARE		0.17

Appendix E

Software Code

The source code for the program described in Chapter 6 is given here. The code is mainly in Visual Basic for Applications, the macro language for Excel 5.0. Included also is a small batch file to run the neural network program with pre-specified input values from a text file, which is included at the end of the code. Not included in the code here is the formatting of the various spreadsheets, and the attachment of procedures to the various control buttons.

```
'Automatically opens Main Menu and clears toolbars, scroll bars etc from interface
Sub Auto_Open()
    Run ("macro1!EchoOff")
    Sheets("Sheet16").Select
    Application.WindowState = xlMaximized
    Application.DisplayFullScreen = True
    Application.DisplayFormulaBar = False
    Application.DisplayStatusBar = False
    With ActiveWindow
        .DisplayHeadings = False
        .DisplayHorizontalScrollBar = False
        .DisplayVerticalScrollBar = False
```

```
        .DisplayWorkbookTabs = False
    End With
    Toolbars(1).Visible = False
    Toolbars(2).Visible = False
    Toolbars(8).Visible = False
    Toolbars(13).Visible = False
    Run ("macro1!EchoOn")
End Sub


'Restore interface with scroll bars etc
Sub RestoreTB()
    Application.WindowState = xlMaximized
    Application.DisplayFullScreen = False
    Toolbars(13).Visible = True
    Toolbars(8).Visible = True
    With ActiveWindow
        .DisplayHeadings = True
        .DisplayHorizontalScrollBar = True
        .DisplayVerticalScrollBar = True
        .DisplayWorkbookTabs = True
    End With
End Sub


'Go to main Menu
Sub GoToM()
    Sheets("Sheet16").Select
    ActiveWindow.DisplayVerticalScrollBar = False
End Sub
```

```
'Go to Worksheet
Sub GoToW()
   Sheets("Sheet15").Select
   ActiveWindow.DisplayVerticalScrollBar = True
End Sub


'Go to Data Entry Sheet
Sub GoToD()
   Run ("macro1!EchoOff")
   Sheets("Sheet2").Select
   Range("A1").Select
   Range("A10").Select
   ActiveWindow.DisplayVerticalScrollBar = True
   Run ("macro1!EchoOn")
End Sub


'Quits Excel
Sub Finished()
   Application.Quit
End Sub


'Creates the input *.net file by:
'1 - clearing data area of old net file
'2 - copy data from DATA FORM into this area
'3 - select whole area and copy
'4 - open NOTEPAD which opens test.txt
'5 - delete everything in old INPUT.NET
'6 - paste new header and data from DATA FORM into INPUT.NET
'7 - saves INPUT.NET
Sub MakeNetFile()
```

```
    Run ("macro1!EchoOff")
    Sheets("Sheet1").Select
    Range("A16:J115").Select
    Selection.Clear
    Sheets("Sheet2").Select
    Range("A10").Select
    Selection.CurrentRegion.Select
    Selection.Copy
    Sheets("Sheet1").Select
    Range("A16").Select
    ActiveSheet.Paste
    Range("A1").Select
    Selection.CurrentRegion.Select
    Selection.Copy
    Range("A1").Select
    tt = Shell("Notepad", 4)
    AppActivate "Notepad - (Untitled)"
    SendKeys "%F O", True
    SendKeys "C:\input.net {Enter}", True
    SendKeys "%E A", True
    SendKeys "%E L", True
    SendKeys "%E P", True
    SendKeys "%F S", True
    SendKeys "%F X", True
    Run ("macro1!EchoOn")
    Sheets("Sheet16").Select
End Sub


'Copies data from TEST.DMP
'Presents it in a usable form
Sub OutputData()
    Run ("macro1!EchoOff")
```

```
Sheets("Sheet3").Select
Range("A1:M100").Select
Selection.Clear
Sheets("Sheet4").Select
Range("A1:M100").Select
Selection.Clear
tt = Shell("Notepad", 4)
AppActivate "Notepad - (Untitled)"
SendKeys "%F O", True
SendKeys "C:\test.dmp {Enter}", True
SendKeys "%E A", True
SendKeys "%E C", True
SendKeys "%F X", True
Sheets("Sheet16").Select
Sheets("Sheet4").Select
Range("A1").Select
ActiveSheet.Paste
Range(Cells(1, 1), Cells(Worksheets("Sheet1").Cells(4, 2).Text * 4, 1)).Select
Selection.Replace What:="Test Pattern ", Replacement:="", LookAt _
   :=xlPart, SearchOrder:=xlByRows, MatchCase:=False
Selection.Replace What:="  Goal: ", Replacement:="", LookAt:= _
   xlPart, SearchOrder:=xlByRows, MatchCase:=False
Selection.Replace What:="Output: ", Replacement:="", LookAt:= _
   xlPart, SearchOrder:=xlByRows, MatchCase:=False
Selection.Copy
Sheets("Sheet3").Select
Range("A1").Select
ActiveSheet.Paste
Sheets("Sheet6").Select
Range("C4:H7").Select
Selection.Clear
Sheets("Sheet4").Select
```

```
    Range(Cells(Worksheets("Sheet1").Cells(4, 2).Text * 4 + 3, 1),
Cells(Worksheets("Sheet1").Cells(4, 2).Text * 4 + 6, 4)).Select
    Selection.Copy
    Sheets("Sheet6").Select
    Range("C4").Select
    ActiveSheet.Paste
    Range("A1").Select
    Sheets("Sheet3").Select
    k = 0
    For i = 1 To Worksheets("Sheet1").Cells(4, 2).Text
      For j = 1 To 4
        k = k + 1
        If j = 2 Then
          Cells(i, "E").Value = Cells(k, 1).Value
        End If
        If j = 3 Then
          Cells(i, "F").Value = Cells(k, 1).Value
        End If
        If j = 4 Then
          Cells(i, "G").Value = Cells(k, 1).Value
        End If
      Next j
    Next i
    Sheets("Sheet5").Select
    Range("D4:F100").Select
    Selection.Clear
    Sheets("Sheet3").Select
    Range("E1").Select
    Selection.CurrentRegion.Select
    Selection.Copy
    Sheets("Sheet5").Select
    Range("D4").Select
```

```
    ActiveSheet.Paste
    ActiveWindow.DisplayVerticalScrollBar = True
    Run ("macro1!EchoOn")
    Range("A1").Select
End Sub


'Calls the Dialog Box
Sub DBox1()
    DialogSheets("Dialog1").EditBoxes("Edit Box 10").Text =
        Worksheets("Sheet1").Cells(1, 2).Value
    DialogSheets("Dialog1").EditBoxes("Edit Box 11").Text =
        Worksheets("Sheet1").Cells(2, 2).Value
    DialogSheets("Dialog1").EditBoxes("Edit Box 7").Text =
        Worksheets("Sheet1").Cells(3, 2).Value
    DialogSheets("Dialog1").EditBoxes("Edit Box 8").Text =
        Worksheets("Sheet1").Cells(4, 2).Value
    DialogSheets("Dialog1").EditBoxes("Edit Box 9").Text =
        Worksheets("Sheet1").Cells(9, 2).Value
    DialogSheets("Dialog1").EditBoxes("Edit Box 12").Text =
        Worksheets("Sheet1").Cells(13, 2).Value
    DialogSheets("Dialog1").EditBoxes("Edit Box 13").Text =
        Worksheets("Sheet1").Cells(12, 2).Value
    DialogSheets("Dialog1").EditBoxes("Edit Box 14").Text =
        Worksheets("Sheet1").Cells(14, 2).Value
    DialogSheets("Dialog1").Show
End Sub


'Update Worksheet
Sub Update()
    Worksheets("Sheet1").Cells(1, 2).Value =
        DialogSheets("Dialog1").EditBoxes("Edit Box 10").Text
```

```
    Worksheets("Sheet1").Cells(2, 2).Value =
        DialogSheets("Dialog1").EditBoxes("Edit Box 11").Text
    Worksheets("Sheet1").Cells(3, 2).Value =
        DialogSheets("Dialog1").EditBoxes("Edit Box 7").Text
    Worksheets("Sheet1").Cells(4, 2).Value =
        DialogSheets("Dialog1").EditBoxes("Edit Box 8").Text
    Worksheets("Sheet1").Cells(9, 2).Value =
        DialogSheets("Dialog1").EditBoxes("Edit Box 9").Text
    Worksheets("Sheet1").Cells(13, 2).Value =
        DialogSheets("Dialog1").EditBoxes("Edit Box 12").Text
    Worksheets("Sheet1").Cells(12, 2).Value =
        DialogSheets("Dialog1").EditBoxes("Edit Box 13").Text
    Worksheets("Sheet1").Cells(14, 2).Value =
        DialogSheets("Dialog1").EditBoxes("Edit Box 14").Text
End Sub


'View Output Data
Sub ViewOutput()
    Run ("macro1!EchoOff")
    Sheets("Sheet5").Select
    ActiveWindow.DisplayVerticalScrollBar = True
    Range("A1").Select
    Run ("macro1!EchoOn")
End Sub


'View Network Statistics
Sub Stats()
    Sheets("Sheet6").Select
    Range("A1").Select
End Sub
```

```
'Copy and Paste
Sub CopyPaste()
   Selection.Copy
   Sheets("Sheet2").Select
   Range("A10").Select
   ActiveSheet.Paste
   Range("A10").Select
   ActiveWindow.DisplayVerticalScrollBar = True
End Sub


'Procedure to run CASCADE
Sub RunCascade()
   dummy = Shell("casrun.bat")
End Sub


'EchoOff
=ECHO(FALSE)
=RETURN()


'EchoOn
=ECHO(TRUE)
=RETURN()
```

'A Batch File to run CASCADE, which is not a Windows based program

CASRUN.BAT

C:\cascadeq.exe < C:\input.txt

'A text file to input some pre-specified values to CASCADE2

INPUT.TXT

C:\input.net

2000

2000

100

1

3

n

n

n

n

y

n

n

References

[1] A.J. Albrecht, *Measuring Development Productivity, Proceedings of the Joint,* SHARE/GUIDE/IBM Application Development Symposium, pp. 83–92, October 1979.

[2] A.J. Albrecht, and J.E. Gaffney, Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation, *IEEE Transactions on Software Engineering*, vol. 9, no. 6, pp. 639–648, 1983.

[3] ASMA (Queensland) Metrics Survey, *State of Metrics in Queensland*, P.O. Box 286, Brisbane, Queensland 4003, February 1994.

[4] ASMA (Victoria), Project Database–Release 5, Australian Software Metrics Association, P.O. Box 1278, Box Hill, Victoria 3128, November 1994.

[5] ASMA (Victoria), Project Database–Release 4, Australian Software Metrics Association, P.O. Box 1278, Box Hill, Victoria 3128, June 1994.

[6] ASMA (Victoria), Project Database–Release 2, Australian Software Metrics Association, P.O. Box 1278, Box Hill, Victoria 3128, June 1993.

[7] R.D. Banker, S.M. Datar, and C.F. Kemerer, A Model to Evaluate Variables Impacting the Productivity of Software Maintenance Projects, *Management Science*, vol. 37, no. 1, pp. 1–18, 1991.

[8] R.D. Banker, R.J. Kauffman, C. Wright, and D. Zweig, Automating Output Size and Reuse Metrics in a Repository-Based Computer-Aided Software Engineering (CASE) Environment, *IEEE Transactions on Software Engineering*, vol. 20, no. 3, 1994.

[9] R.D. Banker, and C.F. Kemerer, Scale Economies in New Software Development, *IEEE Transactions on Software Engineering*, vol. 15, no. 10, pp. 1199–1205, 1989.

[10] E.B. Baum, and D. Haussler, What Size Net Gives Valid Generalization?, *Neural Computation 1*, pp. 151–160, 1989.

[11] C.A. Behrens, Measuring the Productivity of Computer Systems Development Activities with Function Points, *IEEE Transactions on Software Engineering*, vol. 9, no. 6, pp. 648–652, 1983.

[12] B.W. Boehm, *Software Engineering Economics*, Prentice Hall, Englewood Cliffs NJ, 1981.

[13] B.W. Boehm, and P.N. Papacio, Understanding and Controlling Software Costs, *IEEE Transactions on Software Engineering*, vol. 14, no. 10, pp. 1462–1477, 1988.

[14] British Gas, Bang Metric Analysis, Document no. 000763, Process Support, British Gas plc, Dorking, UK, (in [73]), 1991.

[15] F.P. Brooks, The Mythical Man-Month, Addison-Wesley, Reading, 1975

[16] S.D. Conte, H.E. Dunsmore, and V.Y. Shen, Software Engineering Metrics and *Models*, Benjamin/Cummings, Menlo Park, 1986.

[17] J.V.Y. Dam, and P.L. Langbroek, Gebruik van Functiepuntanalyse Vraagt on Beleid, *Informatie*, vol. 34, no. 6, pp. 323–333, 1992.

[18] T. DeMarco, *Controlling Software Projects: Management, Measurement and Estimation,* Yourdon Press, New York, 1982.

[19] B.V. Dasarathy, *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, IEEE Computer Society Press, Los Altimos, 1991.

[20] J.M. Desharnis, Analyse Statistique de la Productivitie des Projets Informatique a Partie de la Technique des Point des Function, *Master Thesis*, University of Montreal, 1989.

[21] J. Diederich, *Personal communication*, Queensland University of Technology, May 1994.

[22] J. Diederich, Explanation and Artificial Neural Networks, *International Journal Man-Machine Studies*, vol. 37, pp. 335-355, 1992.

[23] J.B. Dreger, *Function Point Analysis*, Prentice Hall, Englewood Cliffs, 1989

[24] S. Dutta, and S. Shekhar, Bond rating: A Non-Conservative Application of Neural Networks, *Proceedings of IEEE International Conference on Neural Networks*, vol II, pp. 443–450, 1988.

[25] S.E. Fahlman, *Personal Communication*, Prinicpal Research Scientist, Carnegie-Melon University, 1994.

[26] S.E. Fahlman, The Recurrent Cascade-Correlation Architecture, *Advances in Neural Information Processing Systems 3,* Morgan Kaufmann, Los Altos, pp. 190–196, 1991.

[27] S.E. Fahlman, An Empirical Study of Learning Speed in Back-Propagation Networks, *Technical Report CMU-CS-88-162*, Carnegie-Melon University, 1988.

[28] S.E. Fahlman, and C. Lebiere, The Cascade-Correlation Learning Architecture, *Advances in Neural Information Processing Systems 2,* Morgan Kaufmann, Los Altos, pp. 524–532, 1990.

[29] N.E. Fenton, Software Measurement: A Necessary Scientific Basis, *IEEE Transactions on Software Engineering*, vol. 20, no. 3, 1994.

[30] N.E. Fenton, *Software Metrics: A Rigorous Approach*, Chapman & Hall, London, 1991.

[31] D.V. Ferens, and R.B. Gurner, An Evaluation of Three Function Point Models for Estimation of Software Effort, *IEEE National Aerospace and Electronics Conference – NAECON92*, vol. 2, pp. 625–642, 1992.

[32] G.R. Finnie, and G.E. Wittig, Effect of System and Team Size on 4GL Software Development Productivity, *South African Computer Journal,* no. 11, pp. 18–25, 1994.

[33] G.R. Finnie, G.E. Wittig, and D.I. Petkov, Prioritising Software Development Productivity Factors Using the Analytic Hierarchy Process, *Journal of Systems and Software*, vol 22, no 2, pp. 129–139, 1993

[34] B. Freisleben, Stock Market Prediction with Backpropagation Networks, *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems 5th International Conference IEA/AIE-92,* Paderborn Germany, pp. 451–460, 1992.

[35] Function Point Counting Practices Manual, Release 4.0, *International Function Point Users Group*, Blendonview Office Park, 5008–28 Pine Creek Drive, Westerville, OH 43081–4899, USA, 1994.

[36] V.R. Gibson, and J.A. Senn, 'System Structure and Software Maintenance Performance, *Communications of the ACM*, vol. 32, no. 3, pp. 347–358, 1989.

[37] R.H.M. Gray, B.N. Carey, N.A. McGlynn, and A.D. Pengelly, Design Metrics for Database Systems, *BT Technology Journal*, Vol. 9, no. 4, pp. 69–79, in [73], 1991.

[38] D. Gujarati, *Basic Econometrics*, McGraw-Hill, Singapore, 1979.

[39] M.H. Halstead, *Elements of Software Science*, North Holland, New York, 1977

[40] F.J. Heemstra, Software Cost Estimation, *Information and Software Technology*, vol. 34, no. 10, pp. 627–639, 1992.

[41] J.C. Henderson, and M.E. Treacy, Managing End-User Computing for Competitive Advantage, *Sloan Management Review*, pp. 3–14, Winter 1986.

[42] S. Henry, and D. Kafura, Software Structure Metrics Based on Information Flow, IEEE Transactions on Software Engineering, vol. 7, no. 5, 1981.

[43] J. Hertz, A. Krogh, and R.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley, Redwood City, 1991.

[44] G.E. Hinton, *Neural Networks Seminar*, University of Sydney, February 1993.

[45] M. Hoehfeld, and S.E. Fahlman, Learning with Limited Numerical Precision Using the Cascade-Correlation Algorithm, *IEEE Transactions on Neural Networks*, vol. 3, no. 4, pp. 602–611, 1992.

[46] K. Hornik, M. Stinchcombe, and H. White, Multilayer Feedforward Networks are Universal Approximators, *Neural Networks*, vol. 2, pp. 359–366, 1989.

[47] H. Hruschka, Determining market Response Functions by Neural network Modeling: A Comparison to Econometric Techniques, *European Journal of Operational Research*, vol. 66, pp. 27–35, 1993.

[48] IE, IE-Metrics Knowledge Base, James Martin & Co., Reston, VA, USA, in [73], 1989.

[49] D.R. Jeffery, The Relationship between Team Size, Experience, and Attitudes and Software Development Productivity, *Proceedings: International Computer Software and Applications Conference - COMPSAC87*, pp. 2–8, 1987.

[50] D.R Jeffery, and G.C. Low, Calibrating Estimation Tools for Software Development, *Software Engineering Journal*, pp. 215–221, July 1990.

[51] D.R. Jeffery, Time Sensitive Cost Models in the Commercial MIS Environment, *IEEE Transactions on Software Engineering*, vol. 13, no. 7, pp. 852–859, 1987.

[52] D.R. Jeffery, A Software Development Productivity Model for MIS Environments, *Journal of Systems and Software*, vol. 7, pp. 115–125, 1987.

[53] D.R. Jeffery, G.C. Low and M. Barnes, 'A Comparison of Function Point Counting Techniques', *IEEE Transactions on Software Engineering,* vol. 19, no. 5, pp. 529–532, 1993.

[54] C. Jones, *Personal Communication*, July, 1994.

[55] C. Jones, Assessment and Control of Software Risks, Yourdon Press, Englewood Cliffs, NJ, 1994.

[56] C. Jones, *Applied Software Measurement,* McGraw-Hill, New York, 1991.

[57] C. Jones, *Programming Productivity,* McGraw-Hill, New York, 1986.

[58] C. Jones, The Productivity Report Card, *Software News*, vol. 6, no. 9, p. 19, 1986.

[59] C.F. Kemerer, Reliability of Function Points Measurement, *Communications of the ACM,* vol. 36, no 2, pp. 85–97, 1993.

[60] C.F. Kemerer, An Empirical Validation of Software Cost Estimation Models, *Communications of the ACM,* vol. 30, no 5, pp. 416–429, May 1987.

[61] C.F. Kemerer, and B.S. Porter, Improving the Reliability of Function Point Measurement: An Empirical Study, *IEEE Transactions on Software Engineering,* vol. 18, no. 11, pp. 1011–1024, 1992.

[62] B.A. Kitchenham, *Personal Communication*, NCC, Oxford House, Oxford Road, Manchester M1 7ED, UK, August 1994.

[63] B.A. Kitchenham, Empirical Studies of Assumptions that Underlie Software Cost-Estimation Models, *Information and Software Technology*, vol. 34, no. 4, pp. 211–218, 1992.

[64] B.A. Kitchenham, Software Project Development Cost Estimation, *Journal of Systems and Software*, vol. 5, pp. 267–278, 1985.

[65] J.L. Kolodner, An Introduction to Case-Based Reasoning, *Artificial Intelligence Review*, vol. 6, pp. 3–34, 1992.

[66] R.J. Kusters, M.J.I.M. van Genuchten, and F.J. Heemstra, Are Software Cost-Estimation Models Accurate?, *Information and Software Technology*, vol. 32, no. 3, pp. 187–190, 1990.

[67] K.J. Lang, A.H. Waibel, and G.H. Hinton, A Time-Delay Neural Network Architecture for Isolated Word Recognition, *Neural Networks*, vol. 3, pp. 23–43, 1990.

[68] M.J. Lawrence, Programming Methodology, Organisational Environment, and Progamming Productivity, *Journal of Systems and Software*, vol. 2, pp. 257–269, 1981.

[69] G.G. Lendaris, Professor of System Science and Electrical Engineering, Portland State University, *Personal Communication*, December 1993.

[70] C. X. Ling, Overfitting in Neural-Network Learning of Discrete Patterns, *University of Western Ontario*, 1994.

[71] R.P. Lippmann, and B. Gold, Neural Net Classifiers Useful for Speech Recognition, *Proceedings of the IEEE International Conference of Neural Networks*, pp. 417–425, 1987.

[72] G.C. Low, and D.R. Jeffery, Function Points in the Estimation and Evaluation of the Software Process, *IEEE Transactions on Software Engineering*, vol. 1, no. 1, pp. 64–71, 1990.

[73] S.G. MacDonell, A Comparative review of Functional Complexity Assessment Methods for Effort Estimation, Discussion Paper Series, University of Otago, no. 94/8, ISSN 1172-6024, 1994.

[74] M. Mandischer, Genetic Optimization and Representation of Neural Networks, *Proceedings of the Fourth Australian Conference on Neural Networks,* pp. 122–125, 1993.

[75] J.E. Matson, B.E. Barrett, and J.M. Mellichamp, Software Development cost Estimation using Function Points, *IEEE Transactions on Software Engineering*, vol. 20,.N0. 4, pp. 275–287, 1994.

[76] T.J. McCabe, A Complexity Measure, *IEEE Transactions on Software Engineering,* vol. 2, no. 4, pp. 308–320, 1976.

[77] P. Morris, Charesmatek Metrics Software, *Personal Communication* May 1994.

[78] T. Mukhopadhyay, S.S. Vicinanza, and M.J. Prietula, Examining the Feasibility of a Case-Based Reasoning Model for Software Effort Estimation, *MIS Quarterly*, vol. 16, no. 2, pp. 155–171, 1992.

[79] M.M. Nelson, and W.T. Illingworth, *A Practical Guide to Neural Nets*, Addison-Wesley Reading, 1991.

[80] Neuralyst Version 1.3 User's Guide, Epic Systems Corporation, November 1992.

[81] D.E. Owen, Information Systems Organisations – Keeping Pace with the Pressures, *Sloan Management Review*, pp. 59–68, Spring 1986.

[82] E.A. Patrick, and F.P. Fisher, A Generalised k-Nearest Neighbor Rule for Small Samples Drawn from Uniform Distributions, Information and Control, vol. 16, pp. 128–152, 1970.

[83] K. Power, Leading the World, *Informatics*, pp. 55–56, September 1994.

[84] R.S. Pressman, *Software Engineering: A Practitioners Approach,* New York, McGraw-Hill, 1987.

[85] L.H. Putnam, A General Empirical Solution to the Macro Sizing and Estimating Problem, *IEEE Transactions of Software Engineering*, vol. 4, pp. 345 –360, 1978.

[86] C.V. Ramamoorthy, W-T Tsai, T. Yamaura, A. Bhide, Metrics Guided Methodology, *Proceedings of COMPSAC '85*, Chicago, USA, 1985

[87] R. Rask, Automating Estimation of Software Size During the Requirements Specification Phase, PhD Thesis, University of Joensuu, 1992.

[88] B. Ratcliff, and A.L. Rollo, Adapting Function Popint Analysis to Jackson System Development, *Software Engineering Journal*, pp. 79–84, January 1990.

[89] D.J. Reifer, Asset-R: A Function Point Sizing Tool for Scientific and Real-Time Systems, *Journal of Systems Software*, vol. 11, pp. 159–171, 1990.

[90] F.S. Roberts, *Measurement Theory with Applications to Decision making, Utility, and Social Sciences*, Addision Wesley, Reading MA, 1979.

[91] E.E. Rudolph, Productivity in Computer Application Development, *Auckland University Research Report*, Auckland University, Auckland, 1986.

[92] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, Learning Internal Representations by Error Propagation, *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, Cambridge, MA., MIT Press, vol. 1, pp. 318–362, 1986.

[93] D.E. Rumelhart, and J.L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, Cambridge, MA, 1986.

[94] D.E. Rumelhart, J.L. McClelland, and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, Cambridge, Massachusetts, 1986.

[95] D.E. Rumelhart, B Widrow, and M. Lehr, Applications of Neural Networks in Industry, Business and Science, Unpublished, 1993.

[96] C.J. Sass, and T.A. Keefe, MIS for Strategic Planning and a Competitive Edge, *Journal of Systems Management*, pp. 14–17, June 1988.

[97] R. Schank, 1982. *Dynamic memory: A Theory of Learning in Computers and People.* Cambridge: Cambridge University Press.

[98] N.F, Schneidewind, and H. Hoffmann, An Experiment in Software Error Data Collection and Analysis, *IEEE Transactions on Software Engineering,* vol. 5, no. 3, pp. 276–286, 1979.

[99] E. Schoneburg, Stock Prediction Using Neural Networks: A Project Report, *Neurocomputing*, vol. 2, no. 1, pp. 17–27, 1990.

[100] S. Slade, Case-Based Reasoning: A Research Paradigm, *AI Magazine*, vol. 12, no. 1, pp. 42–55, 1991.

[101] R. Sun, *Integrating Rules and Connectionism for Robust Commonsense Reasoning*, John Wiley & Sons, New York, 1994.

[102] A.J. Surkan, and J.C. Singleton, Neural Networks for Bonding rating Improved by Multiple Hidden Layers, *Proceedings of IEEE International Conference on Neural Networks*, vol II, pp. 157–162, 1990.

[103] C.R. Symons, *Software Sizing and Estimating MkII FPA*, John Wiley & Sons, Chichester, 1991.

[104] C.R. Symons, Function Point Analysis: Difficulties and Improvements, *IEEE Transactions on Software Engineering,* vol. 14, no. 1, pp. 2–11, 1988.

[105] T.P. Szabo, *Personal Communication*, Faculty of Information Technology, Queensland University of Technology, 1994.

[106] C.N.W. Tan, A Study on Using Artificial Neural Networks to Develop an Early Warning Predictor for Credit Union Financial Distress with Comparison to the Probit Model, unpublished research report to the *Australian Financial Institution Commission*, July 1994.

[107] C.N.W. Tan, and G.E. Wittig, 1993. The Study of the Parametric Effect on an Experimental Backpropogation Model. *New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems* , November 1993.

[108] C.N.W. Tan, and G.E. Wittig, 1993. Parametric Variation Experimentation on a Backpropogation Stock Price Prediction Model. *Australian and New Zealand Conference on Intelligent Information Systems ANZIIS-93,* December 1993.

[109] G. Tate, Software Cost Explanation and Estimation, *Australian Conference on Software Metrics – ACOSM 93,* November 1993.

[110] G. Tate, Management, CASE and the Sofwatre Process, *Proceedings of the 12th New Zealand Computer Conference,* Dunedin, 1991.

[111] G. Tate, and J.M. Verner, Approaches to Measuring Size of Application Products with CASE Tools, *Information and Software Technology*, vol. 33, no. 9, pp. 622–628, 1991.

[112] D. Treigueiros, and R. Berry, The Application of Neural Network based Methods to the Extraction of Knowledge from Accounting Reports, *Proceedings of 24th Annual Hawaii International Conference on System Sciences*, vol IV, pp. 137–146, 1991.

[113] J.M. Verner, G. Tate, B. Jackson, and R.G. Hayward, Technology Dependence in Function Points: A Case Study and Critical Review, *Proceedings: 11th International Conference on Software Engineering*, pp. 375–382, 1989.

[114] I. Vessey, On Program Development Effort and Productivity, *Information and Management*, vol. 10, pp. 255–266, 1986.

[115] C.E. Walston, and C.P. Felix, A Method of Programming Measurement and Estimation, IBM Systems Journal, vol. 10, no. 1, pp. 10–29, 1977.

[116] H. White, Economic Prediction Using Neural Networks: The case of IBM Daily Stock Returns, *IEEE Conference on Neural Networks*, vol. 2, 1988.

[117] M.L. Wilson, The Measurement of Usability, in *Entity-Relationship Approach to Systems Analysis and Design*, P.P. Chen (ed.), North-Holland, pp. 75–101, 1980.

[118] G.E. Wittig, Processes and Knowledge Structures for case-Based Reasoning, *Working Paper 1993-3-81/B*, Bond University, March 1993.

[119] G.E. Wittig, An Analysis of 4GL Software Development Productivity, *Masters Thesis*, University of Natal, 1991.

[120] G.E. Wittig, and G.R. Finnie, Using Artificial Neural Networks and Function Points to Estimate 4GL Software Development Effort, *Australian Journal of Information Systems*, vol. 1, no. 2, pp. 87–94, 1994.

[121] G.E. Wittig, and G.R. Finnie, Software Design for the Automation of Unadjusted Function Point Counting, *International Federation for Information Processing Conference–IFIP TC8AUS Conference*, May 1994.

[122] G.E. Wittig, and G.R. Finnie, A Study of Software Development Effort Estimation Model Performance, *Working Paper 1994-3-109/B*, Bond University, April 1994.

[123] G.E. Wittig, and G.R. Finnie, Software Development Productivity Variations in Function Point Research Data, *Working Paper 1994-3-108/B*, Bond University, April 1994.

[124] Y. Yoon, G. Swales, J.R. Margavio, and T.M. Margavio, A Comparison of Discriminant Analysis versus Artificial Neural Networks, *Journal of the Operational Research Society*, vol. 44, no. 1, pp51–60, 1993.